



**Calhoun: The NPS Institutional Archive**

---

Theses and Dissertations

Thesis Collection

---

1985

Prototyping with data dictionaries for requirements analysis.

Noel, Alan F.

---

<http://hdl.handle.net/10945/21513>



Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

**Dudley Knox Library / Naval Postgraduate School**  
**411 Dyer Road / 1 University Circle**  
**Monterey, California USA 93943**

<http://www.nps.edu/library>







DI  
NA  
MONT

DUDLEY KNOW LIT  
NAVAL FC  
MONT  
3







# NAVAL POSTGRADUATE SCHOOL

Monterey, California



## THESIS

PROTOTYPING WITH DATA DICTIONARIES  
FOR REQUIREMENTS ANALYSIS

by

Alan F. Noel

March 1985

Thesis Advisor:

Daniel R. Dolk

Approved for public release; distribution is unlimited

T223864





REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Prototyping With Data Dictionaries For Requirements Analysis		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis March 1985
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Alan F. Noel		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		12. REPORT DATE March 1985
		13. NUMBER OF PAGES 198
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution is unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Data Dictionary, Software Prototyping, Requirements Analysis		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The analysis of software system requirements is to develop a complete, consistent and unambiguous statement for what the software will do, but not how. Failure to correctly elicit requirements is cause for difficult and expensive correction efforts during later phases of system development. An alternative approach to interviewing and textual document preparation is the use of a functionally limited model, a prototype. (Continued)		



ABSTRACT (Continued)

This thesis will examine the feasibility of using prototypes accompanied with data dictionaries as a more effective means of communicating with users and, therefore, more correctly eliciting requirements. It will include development of a relational based, prototype data dictionary using the dBASE II database management system. The dictionary is for use by the Deputy Chief of Staff for Plans, U. S. Army Military Personnel Center, Alexandria, Virginia.

Approved for public release; distribution is unlimited.

Prototyping With Data Dictionaries  
For  
Requirements Analysis

by

Alan F. Noel  
Major, United States Army  
B.A., University of Florida, 1973

Submitted in partial fulfillment of the  
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL  
March 1985



## ABSTRACT

The analysis of software system requirements is to develop a complete, consistent and unambiguous statement for what the software will do, but not how. Failure to correctly elicit requirements is cause for difficult and expensive correction efforts during later phases of system development. An alternative approach to interviewing and textual document preparation is the use of a functionally limited model, a prototype.

This thesis will examine the feasibility of using prototypes accompanied with data dictionaries as a more effective means of communicating with users and, therefore, more correctly eliciting requirements. It will include development of a relational based, prototype data dictionary using the dBASE II database management system. The dictionary is for use by the Deputy Chief of Staff for Plans, U.S. Army Military Personnel Center, Alexandria, Virginia.

## TABLE OF CONTENTS

I.	INTRODUCTION . . . . .	10
II.	PROTOTYPING FOR SOFTWARE REQUIREMENTS . . . . .	14
	A. DEFINITION OF PROTOTYPING . . . . .	14
	B. ADVANTAGES OF PROTOTYPING . . . . .	16
	C. DISADVANTAGES OF PROTOTYPING . . . . .	19
	D. APPLICATIONS AND GUIDANCE FOR PROTOTYPING . . . . .	20
	E. TOOLS FOR PROTOTYPING . . . . .	23
	F. DATA DRIVEN PROTOTYPING . . . . .	26
	G. RELATION BETWEEN PROTOTYPING AND DATA DICTIONARIES . . . . .	28
III.	DATA DICTIONARIES . . . . .	30
	A. DATA CENTERED SYSTEM ANALYSIS . . . . .	30
	B. METADATA . . . . .	32
	C. DEFINITION AND OBJECTIVES OF A DATA DICTIONARY . . . . .	40
IV.	A DATA DICTIONARY BASED ON THE RELATIONAL MODEL . . . . .	45
	A. INTRODUCTION . . . . .	45
	B. SYSTEM STRUCTURE AND USER INTERFACE . . . . .	49
V.	CONCLUSIONS . . . . .	62
	APPENDIX A: USER MANUAL . . . . .	64
	A. INTRODUCTION . . . . .	64
	B. INITIALIZING THE DEMONSTRATION SYSTEM . . . . .	64
	C. INITIALIZING THE INSTALLED DICTIONARY . . . . .	65
	D. MAIN MENU . . . . .	66
	E. ADD SUBSYSTEM . . . . .	67



F. CHANGE SUBSYSTEM . . . . .	68
G. DELETE SUBSYSTEM . . . . .	70
H. QUERY SUBSYSTEM . . . . .	71
I. REPORTS SUBSYSTEM . . . . .	73
J. TERMINATING A SESSION . . . . .	77
APPENDIX B: DATA DICTIONARY SYSTEM'S DATA ELEMENTS . . .	78
APPENDIX C: DATA DICTIONARY SOURCE LISTING . . . . .	93
LIST OF REFERENCES . . . . .	194
INITIAL DISTRIBUTION LIST . . . . .	198

## LIST OF TABLES

I	The Grouping of Meta-Entities . . . . .	39
---	---	----

## LIST OF FIGURES

2.1	Steps in Traditional System Development . . . . .	17
2.2	System Development Using Prototyping . . . . .	18
2.3	Integrating Function of A Data Dictionary . . . . .	29
3.1	Hierarchy of Metadata Entities . . . . .	35
4.1	Bachman Diagram of Entities . . . . .	48
4.2	Hierarchical Chart of Software . . . . .	50
4.3	Example of a Forward Retrieval . . . . .	51
4.4	Sample Output From a Forward Retrieval . . . . .	51
4.5	Example of a Backward Retrieval . . . . .	52
4.6	Output From a Backwards Retrieval . . . . .	53
4.7	Example of FORAL Menu . . . . .	53
4.8	Example of Output From FORAL Query . . . . .	54
4.9	Example of BAKAL Program . . . . .	54
4.10	Example of Output from BAKAL . . . . .	54
4.11	Example of Menu for FORCAT Program . . . . .	55
4.12	Example of Output From FORCAT . . . . .	55
4.13	Example of Menu for BAKCAT Program . . . . .	56
4.14	Example of Output for BAKCAT Program . . . . .	56
4.15	Example of Report Output . . . . .	57
4.16	First Example of Demonstration Data . . . . .	59
4.17	Second Example of Demonstration Data . . . . .	60
A.1	Example of a Forward Retrieval . . . . .	72
A.2	Example of Forward Query Output . . . . .	73
A.3	Example of a Backward Retrieval . . . . .	74
A.4	Output From a Backwards Query . . . . .	74
A.5	Example of Report Output . . . . .	76



## ACKNOWLEDGEMENTS

I extend my heartfelt thanks to Ms. Brenda Morillo whose patience, careful proofing and page numbering was of invaluable assistance.

## I. INTRODUCTION

The analysis of requirements has been, and continues to be, a source of difficulty in developing software. The impact of improper analysis is far reaching. Consequently, considerable effort has been put into developing methodologies to elicit and accurately capture what is needed and wanted. The traditional strategy for conducting requirements analysis has been to interview users individually and then to document, in ordinary English, the users' needs. The result of this process is a large tome of requirements that is seldom understood in its entirety. However, requirements analysis is the foundation effort for any software system and therefore of particular importance for achieving project success. The foundation is provided by uncovering the flow and structure of data.

The software to be built is described by several completion of several actions. Design constraints must be identified to bound the problem and clarify its scope. Functions and data must be described to permit an understanding of the essence of the system to be automated. Interface details for hardware, software, human beings and procedures must be developed to integrate the software into the encompassing system or organization. Important to describing the software is establishing and maintaining communications with both the eventual user and the originating requestor. The information flow and structure becomes a tool that leads to an overall representation of the software which will be developed. Thus, the design process is fed the output of the requirements analysis effort. The continuity of the effort and the quality of the design are heavily dependent on the requirement specification. [Ref. 1]

There are several problems with the requirement analysis effort as exercised in the traditional software life cycle. First, users' needs may change significantly during development. This can occur through a natural evolution of the organization or through interaction with the software development effort. Secondly, the user has no system during the period between the solicitation of requirements and delivery of the completed system. The user has no clear picture of what he will be getting because he has only been dealing with abstract terms and representations, nothing that actually behaves like the real system. [Ref. 2]

The reasons for some of these problems are known. The people involved do not share the same frames of reference because analysts and end-users come from different cultures. An analyst can be viewed as a missionary steeped in computer technology. An end-user is only concerned with getting the job done within his particular business area. The two usually do not understand each others' area of expertise and a great deal of learning, particularly for the analyst, must take place. The product resulting from the interaction of an analyst and a user is often a document of which too much is expected. The medium of a textual document may capture many relevant details, but it may not in turn communicate them with sufficient clarity to permit true understanding. A specification document is often too long; contains technical terms the user does not know; contains words with different meaning to users and analysts and often also contains trivia and motherhood statements which sound good but offer no real benefit. A textual specification can be particularly lacking regarding interactive systems because a document cannot capture the dynamic quality of the user interaction.

Nevertheless, the textual document as a requirement specification is the basis for all follow on work. This

work cannot be correct if its basis is inaccurate. Despite users not always understanding the specifications, they are often coerced into signing and approving the document because the developers refuse to go forward without a signature. The developers feel it is necessary to have user approval and the requirements frozen to prevent having to make changes which increase system cost further along in the development process. However, the freezing of requirements specifications causes apprehension for both users and developers. Developers either intuitively feel or know from experience that the act of providing what an end-user needs, and the follow on design and coding effort, changes his perception of those needs. The solution to a problem can change the problem [Ref. 3]. The essence of the problem of inadequate requirement specifications is the incomplete communications between the user and the analyst. The evidence that inadequate communications is still a problem lies in the fact that software development projects still suffer from poorly defined specifications despite the considerable effort that has been expended in developing and using rigorous methodologies. Some observers feel the failure rate is just as high as it was 15 years ago and that most development is a trial and error process [Ref. 4].

Requirements specifications can take one of three forms. The traditional approach is to develop a textual list of "shall statements" the system must fulfill. A major disadvantage of the textual approach is the psychological distance from what the user will eventually receive. A document does not convey a realistic sense of how the system will fit the user's needs. This is especially true for interactive systems. Textual specifications take a static view in attempting to capture or freeze the situation. Most users cannot fully describe their current requirements and even fewer can identify their future needs. A more recent



innovation is the use of an interpretive model. An example of this approach is the Structured Analysis Design Technique which uses a set of graphical representations. Other interpretive modeling techniques are the use of data flow diagrams, flowcharts and Nassi-Schneidermann charts. These approaches, although an improvement over the textual, still do not adequately communicate on line-interaction. Using more detailed graphics places more burden on the users than it helps. However, users know what would be useful if they saw it [Ref. 5]. A useful approach is to develop a working model, a prototype. The development of higher level languages and the lessening cost of hardware have recently given rise to a requirements analysis approach which offers promise of assisting in the communications problem. This technique is prototyping.

Chapter 2 is devoted to explaining the concept of prototyping. A key to successful prototyping, if not all systems analysis, is the capture of data about the data in some form. Data dictionaries are the means most often used for organizing this "metadata". Chapter 3 will discuss metadata and data dictionaries and their potential role in the system analysis and design process. By explaining prototyping and data dictionaries we will show the two as an inseparable pair which are an important adjunct to the system development process. The culmination of this thesis is chapter 4 wherein we develop a general, relational-based design of a data dictionary for use with prototyping for requirements analysis. This dictionary will be implemented using dbase II, a micro-oriented relational database management system.

## II. PROTOTYPING FOR SOFTWARE REQUIREMENTS

### A. DEFINITION OF PROTOTYPING

Webster's dictionary defines a prototype as one of three possible things:

1. An original or model after which anything is formed
2. The first thing of its kind
3. A pattern, an exemplar, an archetype

The second definition is probably the most relevant to this discussion because prototypes are being used in data processing as a first attempt at a design which is then extended or enhanced. In general systems development, a prototype is known as

"...a partially complete functional model of a target system whose purpose is to provide a better understanding of the target system's requirements" [Ref. 6].

A software prototype is characterized by the following features. It is a working system, although of limited capability, rather than just an idea on paper. A prototype may become, after iterative enhancement, a production system. Its original purpose is to test out assumptions about requirements and/or system design architecture. A prototype is created quickly. This has become possible only in recent years with more powerful languages which are less procedurally oriented. Some would argue however, that prototyping was the way software was developed before the advent of functional decomposition and the system development life cycle which is generally accepted and used today.

"In the early days of software development writing programs was the thing to do. After an explanation of the problem, a period of questions and answers, and research into the nuts and bolts of a method, the programmer began his or her work. Starting with that portion of the problem that was well understood, lines of FORTRAN, COBOL or ALGOL would begin to appear. As time passed additional portions were coded until the entire program was complete. Design was conducted implicitly, if at all!" [Ref. 7]

A prototype should be inexpensive to build, at least less than it would cost if a conventional high level language were used. Indeed, prototyping in data processing originated only recently because until now programming a prototype was just as costly as programming the working system [Ref. 8]. The important point is to get something running soon to establish effective communications with the user without the use of extravagant resources. The follow on development of a prototype is an iterative process in which improvements are made in small increments as the user and developer work together and discover new requirements [Ref. 9].

Mitchell Spiegel, formerly of Wang Laboratories, explains the prototyping approach as:

"...a process of modeling user requirements in one or more levels of detail, including working models. Project resources are allocated to produce scaled down versions of the software described by requirements. The prototype version makes the software visible for review by users, designers and management. This process continues as desired, with running versions ready for release after several iterations." [Ref. 10]

Traditional management information system development follows a series of steps (see Figure 2.1). Prototyping is considered as an adjunct activity to the specification of requirements (see Figure 2.2). The results of prototyping are input to the steps following requirements analysis, but may or may not be used actively in those steps.

Three alternatives are possible for the disposition of a prototype. Having served the purpose of facilitating

requirements definition the prototype can be discarded. The lessons learned are transferred to a requirements specification and development of the system then follows the traditional system development cycle. A second alternative is to maintain the prototype and run it in parallel with the developing target system. The benefit of this choice is the availability of a partially functional system to the users until the fully capable system can be delivered. Lastly, the prototype can continue to be iteratively refined to become the target system. This alternative is usually only feasible in management information systems as opposed to embedded software, such as weapon control systems [Ref. 11]. Some would argue that the term prototype should only be applied to those instances where the system is to be thrown away, otherwise it is more appropriate to term the activity as evolutionary development [Ref. 12]. The purpose of the distinction is to insure that only the right decisions are kept and transferred to the requirements specification while the mistakes, especially those causing poor execution efficiency, are left behind with the prototype.

## **B. ADVANTAGES OF PROTOTYPING**

There are several advantages associated with the use of prototyping. The primary argument for using prototypes is that the specifications which result are more complete because users can evaluate a prototype better than they can evaluate written specifications. In using the prototype the users encounter and think about problems they will have with the production system [Ref. 13]. Another benefit is that prototyping allows and encourages users to change their minds about what they want. Consequently, the eventual system is better customized and therefore more useful. This relates to the idea that the development process itself



```
graph TD; A[Feasibility Study] --> B[Requirements]; B --> C[Product/ Preliminary Design]; C --> D[Detailed Design]; D --> E[Coding]; E --> F[Integration]; F --> G[Implementation]; G --> H[Operations and Maintenance];
```

Feasibility Study

Requirements

Product/ Preliminary Design

Detailed Design

Coding

Integration

Implementation

Operations and Maintenance

Figure 2.1 Steps in Traditional System Development

changes the users' perceptions of what is possible, increasing their insights into the development process and affecting it. In fact, this idea is a loose corollary of the Heisenberg Uncertainty Principle which states that any system development activity inevitably changes the environment out of which the need for the system arose [Ref. 14]. Some argue that prototyping eliminates the surprises at the end of the traditional development process because specifications, some being erroneous, haven't been frozen and pursued without correction. Another argument is that prototyping shortens the development cycle, but experience to date has found this not always to be the case and that prototyping only more clearly defines what has to be done. Of importance to those suffering with technically oriented data processing personnel who have difficulty relating to people is the benefit that prototyping does not

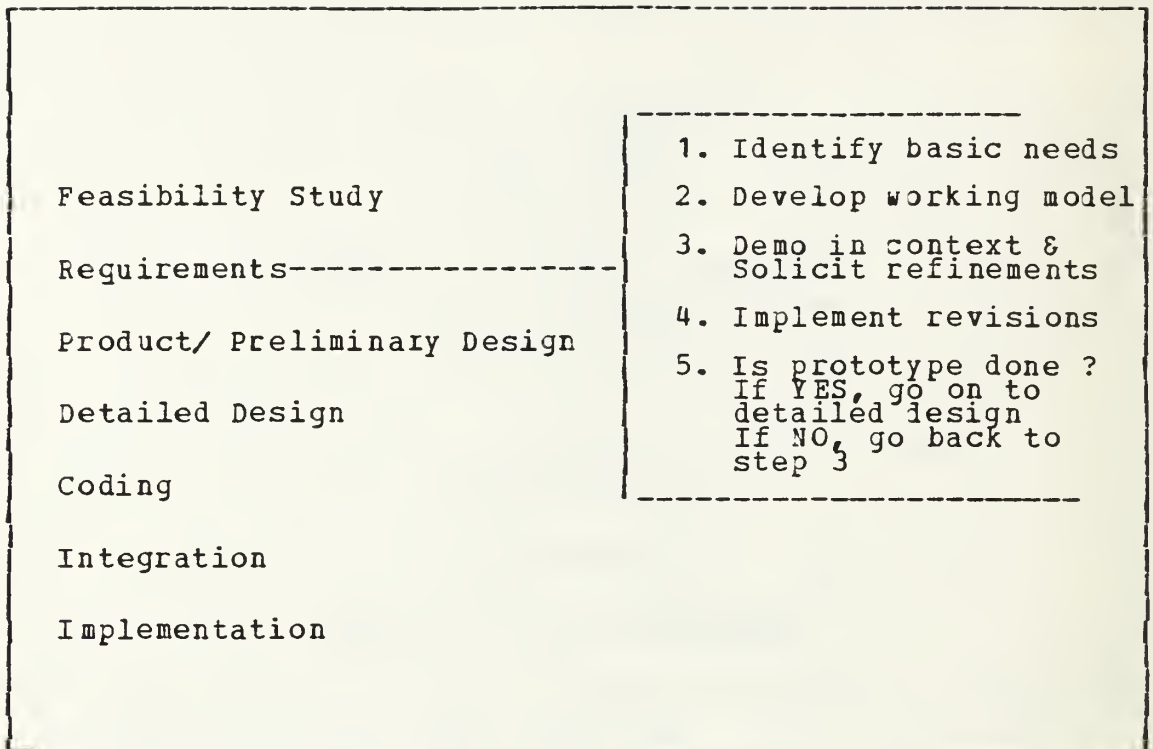


Figure 2.2 System Development Using Prototyping

require interviewing skills to the same degree as a traditional method of defining requirements. Also, because prototyping requires spending more time with users, the systems analysts are forced to become more user oriented and aware of the user's environment.

The emphasis on the user is the key issue here and further benefits are possible. The availability of a prototype enables the user to evaluate the human interfaces in practice and suggest changes. The user gets a more personalized system, regardless of the efficacy of the changes suggested. Thus the user is more willing to accept the eventual system having the perception that he truly affected its shape.

A working prototype enables the developer to evaluate the users' performance with the human interface and to

modify it to minimize errors. Experimentation is facilitated by implementing alternative interfaces without having to suffer the tedium of drawing specification graphics of one sort or another.

It is possible to generate human-machine dialogues and to make minor changes quickly.

The existence of a prototype gives the user a more immediate sense of the proposed system and encourages him to think more carefully about both the needed and the desirable characteristics of the system. It is easy to write statements in a requirements document which say "the system shall do x" and "the system shall be capable of y". However, both the developer and the user get a more realistic feeling for the effort and cost of a feature when they must actually add it to a working model. Thus, the eventual model better represents what is feasible than a document with a series of "shall statements". It is particularly significant to note that in a typical development effort there is a long time between analysis and construction. The user only has a limited idea of what the system will do in such a case. Therefore, it's difficult for the user to be specific when dealing only with abstract terms. Even though the functionality of a prototype system is minimal the user is forced to think more carefully about the task being automated. This should produce a more accurate understanding of the problem [Ref. 15]. The biggest argument for prototyping is that, unlike traditional methods, it builds an effective bridge across the communications gap between the user and the analyst.

### C. DISADVANTAGES OF PROTOTYPING

Prototyping has some decided disadvantages as well. Prototyping makes it difficult to plan resource use because

a clear path of what you will be doing and when isn't provided. It also makes it difficult to decide whether to enhance an old system or build a new one. Analysts and users can become bored after the nth iteration of the prototype. In using the traditional development process there are specific requirements which, when met by proof of validation, clearly mark the job as complete. Because the prototype is changing continually, it creates a problem keeping users abreast of the current version.

Prototyping can cause a reduction in discipline for proper documentation and testing. Because there is less emphasis on hard thinking and "desk checking" there is a greater chance of missing a basic problem which could negate assumptions essential to the design being developed. For large systems it is often not clear how the problem can be divided up so it can be prototyped a piece at a time until a traditional requirements study has been made. Also there is the chance users may become so happy with the prototype they consider it a functional product and want the data processing people to start work on something else. This can create a higher initial cost for the requirements phase and possible loss of the distinction between this phase and the design phase.

A study using the ACT/1 software package for prototyping showed increased needs for computing resources. If the productivity gained from using prototyping doesn't offset the cost of the increased computing power, then the prototyping approach is serving at a disadvantage.

[Ref. 16]

#### D. APPLICATIONS AND GUIDANCE FOR PROTOTYPING

Prototypes can be usefully applied to several different tasks associated with requirements analysis. These tasks are as follows [Ref. 17] :



1. A prototype can be used to discover the optimum structure for the data about data (metadata). This relates to the engineering use of prototypes as an experimental model from which to learn.
2. The prototype can serve as a basis for bottom-up integration.
3. They can be used to define macros that compute or derive new data from stored data. Abstracting in this manner is a process of deriving macros from a database. The database is enhanced having these macros incorporated into it.
4. Prototypes can be used to validate domains. This is particularly important if there is going to be an active data dictionary which will assist in ensuring data integrity.
5. Prototypes can be used to evaluate both update and retrieval efficiency.
6. They can be used to improve the physical database design.
7. They can be used to anticipate changes. This means to conduct experiments and step through what the user will do with the system to gain insight into how his use will foster new requirements, be they modifications or enhancements.
8. They can be used to facilitate implementation and acceptance of a new system.

If any of the items listed above seems sufficient to warrant a prototype then several conditions should exist to make prototyping an appropriate choice. These conditions are as follows:

1. The user's requirements can only be defined with much study.
2. The user's needs are expected to change often.
3. The user is willing to cooperate with the prototyper in answering questions.
4. Users are willing to accept the prototype as an adequate and temporary solution until an optimized and tested system can be delivered.
5. The area of study is well defined. Although in depth aspects of a problem aren't known, the width or scope is confined to a particular business area. Prototypes aren't appropriate for developing information systems strategies for an entire enterprise.
6. The user has a limited set of objectives. Projects of large size with many different applications areas are not appropriate choices for use of the prototyping methodology. The project must first be functionally decomposed to identify specific business areas.
7. A data dictionary is available for use. Preferably an automated, easy-to-use dictionary is available to facilitate easy recording of specifications, data entities, procedures and whatever else is necessary to model the information needs of the user [Ref. 18].
8. Complete honesty exists between the user and the prototyper. If the user isn't satisfied, he should say so and be specific. Further, the prototyper should admit ignorance and misunderstanding. If the answer is no to a particular feature because it isn't technically feasible, it should be so stated and explained, not ignored. The developer must be willing to interpret user difficulties as requiring re-examination of the application, rather than

"educating" the user, i.e. converting the user to the developer's concept [Ref. 19].

Regarding the last point above, technical people are generally deeply involved with their own inventive processes and the reasons for their choices, and therefore are not inclined to consider other roads which also admit acceptable solutions.

## E. TOOLS FOR PROTOTYPING

Prototyping to define system requirements is a methodology. A methodology is a combination of tools and techniques employed within an organizational and managerial framework that can be consistently applied to successive information system development projects [Ref. 20]. A requirements analysis tool is any set of procedures, manual or automated, that guides an analyst through requirements specification. A manual tool is a well defined technique usually with a mode of graphic representation. An automated technique is often a specification language that combines keyword indicators with a natural language feature. There is also sometimes a specification language which, when fed to a processor, produces a requirement specification and a set of diagnostic reports concerning the consistency and organization of the specifications provided. [Ref. 21]

The problem with currently available manual and automated systems is the necessity of the user to learn some technical features to use the tool. An argument against these techniques is that the end-user should not be burdened with learning about the data processing field. Thus, some would argue it's beneficial to have the user interact only with that which will become specifications for, if not part of, the eventual system. A categorization of tools for prototyping has been made and offers a means of examining

what is available for use. The basic requirement for any of these items is that they can be used for prototyping if they allow designers and programmers to create a working system in a short period of time.

The first category of tools is data manipulation systems or data base management systems which can interface with a nonprocedural language to do data manipulation rather than just screen presentation. Such products as RAMIS, FOCUS and NOMAD are examples of this category of tool [Ref. 22]. Also in this category are relational data base management systems which some feel are particularly suited for prototyping.

Relational data base management systems offer a high level of ease of use and data independence, both of which are advantageous for prototype development. The ease of use comes from the simple table data structures and the query languages associated with most relational systems. Data independence permits simple changes to logical or physical design without affecting existing programs [Ref. 23]. Relational data base management systems in this instance are being used as applications generators. Other examples of applications generators are report generators, graphics packages, statistical packages and software for screen layout. These items work at a higher level than another category of prototyping tool, program generators.

Program generators are software packages which, when given specifications, produce source listings, usually in a high order language such as COBOL, capable of being compiled and executed. Although not specifically oriented in the requirements definition area, they do provide assistance to requirements definition if it is necessary for the prototype to do actual data processing. These are particularly helpful to programmers who must enhance the prototype into a production system. Such program generators are not in



proliferation at the current time, but a sufficient number exists to make them usable in some development efforts.

Another category of prototyping tool is libraries of reusable code. If a library of previously written routines, especially screens, exists which can be readily implemented to demonstrate a system's limited capabilities then the prototyper can easily pick and choose. This saves considerable effort and speeds along the process. However, such a library must be well organized so that the individual building the prototype can readily identify modules which suit his needs and can move these modules into the developing prototype with relative speed. This implies the code used has been well documented and generalized to be applicable beyond its original intent. The prototyper must be able to maintain momentum with users and this will not happen if he must stop to develop a detailed program.

[Ref. 24]

Hardware as well as software should be considered in developing prototypes. The use of a microprocessor offers some unique benefits as a prototyping tool. A micro can be beneficial because it offers a cheaper alternative when considering the cost of acquiring a fourth generation language or a data base management system for a mainframe. A micro is portable and therefore can be taken to a user's area where currently no terminals exist. The micro can be left for the users to play with and discuss at their convenience without the possible dampening result of having a data processing person around. This way the end users are not imposed on by disruptive interviews. Finally, having to actually implement the features provides system builders better understanding of what the user wants and avoids the situation of just adding verbiage to a requirements specification. [Ref. 25]



## F. DATA DRIVEN PROTOTYPING

There exists a categorization of prototyping techniques which divides all efforts into 1 of 2 possibilities. The first category is termed applications-oriented and describes the situation when the focus is on transaction modeling and simulation technology. Most people who belong to this school believe prototypes should be thrown away rather than carried forward and evolved. The other school, the data driven category, believes prototypes should evolve, extending the architecture of the business in a process akin to learning.

The thought is that requirements analysis should be an open system which learns from its experiences. It holds that freezing requirements makes for a closed system. A closed system is not responsive to the user and therefore doesn't reflect those desires which will make the system usable.

The data driven prototyping technique was created around an abstraction approach. The analysts and users should abstract from an entity base rather than decomposing from a functional base. The data driven school of thought contends that functional decomposition is not realistic because its use erroneously assumes requirements can be precisely determined before system construction is attempted. The prototyping approach is considered valid because it operates on the more realistic assumption that precise requirements are not always definable until after construction is started. Some argue that the top down design techniques, another way of referring to functional decomposition, are imperative in nature; that is, they force us to concentrate on the operations in the solution space, with little regard for discerning data structures.

The major difference between application and data driven prototyping is that the latter emphasizes a dominant role and value for data. Data prototypers believe entities are the essential elements of learning and that they transcend individual system boundaries. The data driven school contends that functional decomposition is not realistic because its use erroneously assumes requirements can be precisely determined before system construction is attempted.

The prototyping approach is considered valid because it operates on the more realistic assumption that precise requirements are not always definable until after construction is started. They assert that there is a data life cycle distinct from the traditional systems life cycle. Further, they believe the degree of integration of systems is a function of how much data are shared by the systems. From these concepts it is the job of the prototyper to discover and manage the integrating data in the most efficient and least expensive manner. The purpose of the data driven approach is to develop and maintain shared data bases (open systems) that are constantly being extended to accommodate new information requirements. The data bases should facilitate the extraction and synthesis of data into useful information from a static set of data structures.

Several steps are required to apply the data driven approach to a particular problem.

1. Operational reviews are conducted to evaluate the "as is" environment.
2. Conceptual design is conducted to define the metadata structure at the center of what will evolve.  
Scenarios are built in this step to describe those functions for changing or retrieving the metadata.
3. The metadata is normalized in the data design phase where the physical organization of the data is built.

4. Heuristic analysis then follows where requirement scenarios are exercised against the metadata structures using real data values.
5. Environmental testing is conducted where programs are built to support data acquisition and population of the data bases.
6. Quality control is applied via use of performance modeling to ensure the system not only reflects the user's desires but is processing in a realistic fashion.

Proceeding through these steps requires several tools such as items for information modeling, activity modeling, project control, a relational data base management system and a data dictionary. [Ref. 26]

#### G. RELATION BETWEEN PROTOTYPING AND DATA DICTIONARIES

The increasing complexity of computer systems and the growing demand of management that computer systems justify their cost and support decision-making calls for an integrated tool that is capable of recording and processing information. This information describes both the structure and usage of all the enterprise's data and functions. A data dictionary is a tool that, if integrated with prototyping and the traditional methodologies, can meet the demands of complex systems and management [Ref. 27].

Prototyping has proven so successful that some computer service companies have been developing packages to provide prototyping off the shelf. Most of these packages assume the existence of a fully descriptive data dictionary or else require the establishment of one [Ref. 28]. Some feel that the eventuality of extensive networking and distributed processing coupled with failure to properly elicit needs makes it imperative that data dictionaries be integrated into requirements analysis efforts [Ref. 29].

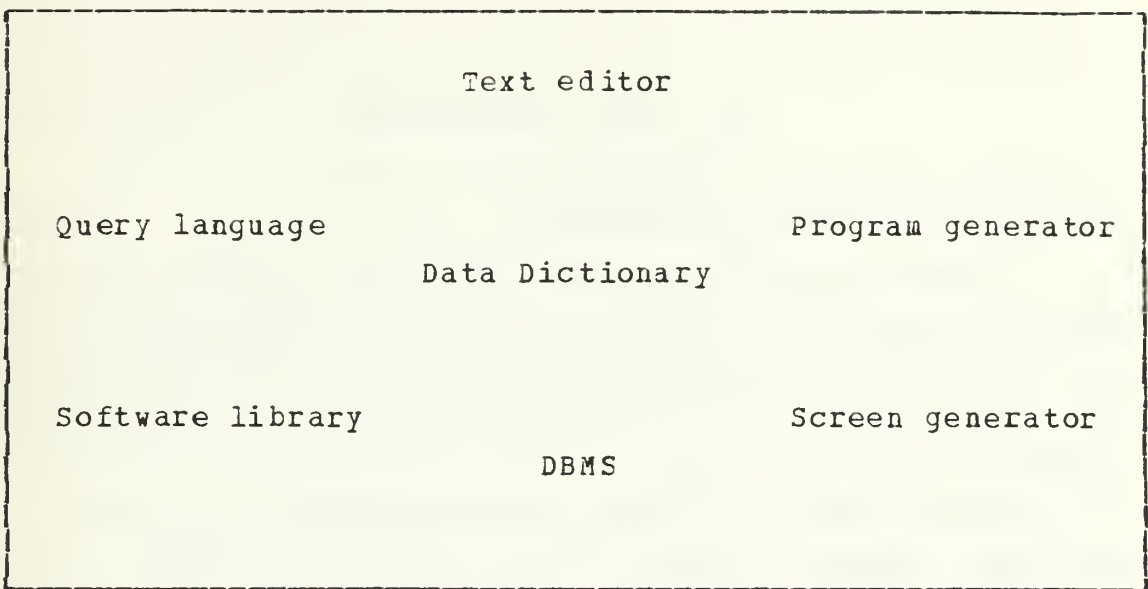


Figure 2.3 Integrating Function of A Data Dictionary

The data dictionary that results from requirements analysis can be a two-edged sword. It can become both a source and receptacle for the prototyping effort. Without a data dictionary of some type, preferably one that can be used as an adjunct to a software library, the analysis task is impossible. The dictionary can be used as a source of information necessary for prototyping since it can define entities, attributes and relationships at a conceptual level. A data dictionary can contain the output of prototyping in an organized, easy to manipulate database instead of using an unwieldy written document [Ref. 30]. In essence, a data dictionary should be at the center of the effort and serve to integrate all the conceptual output produced by the tools being used.

### III. DATA DICTIONARIES

#### A. DATA CENTERED SYSTEM ANALYSIS

An understanding of the role data dictionaries play in software prototyping first requires an understanding of the underlying philosophy regarding their use. This philosophy is a data centered view of system analysis as opposed to a process centered approach.

Data centered system analysis treats the data as a separate resource. Processes are still considered important, but the target of all processing activity is data. An aspect of this approach is a perception of data processing as a succession of changes to data. A process is viewed as a series of data changes. The data centered approach contends that, in the long run, designing a stable, well-documented and largely non-redundant structure of data provides a simpler and cleaner form of data processing than embedding separately designed data structures into many different processes or applications programs.

Several advantages accrue from the data centered approach. The data centered approach helps avoid problems with file proliferation, maintenance, data redundancy and data inconsistency. Once the appropriate data bases are in place, some types of applications can be created quickly with high level data base languages. Individual users having direct access to the data bases can often create their own reports and applications, thus avoiding the slow steps of formal system analysis. Assembling data in different ways from the current inventory is not possible without a data dictionary. Data administrators, programmers and users cannot realistically be expected to have



sufficient knowledge of the logical organization of data bases to readily pick and choose data as needed to derive applications.

There are distinct disadvantages to the data centered approach. The data bases used must be well designed, yet the technology of good data base design is often not understood. The data centered approach implies the sharing of data which requires different management from that applied to a collection of separate files. There are conversion problems in moving from the traditional process centered to the data centered approach. Some of these are human behavioral problems caused by users losing control of data they previously "owned". The cost of conversion to a data base environment can be enormously expensive. The data centered approach is often more difficult to manage in its early phases, because of pressures to adopt quick alternate solutions or to maintain the status quo. Once set up, however, the data centered approach is easier to manage than a process centered one. Management must understand these issues clearly for the approach to work.

A key to controlling data bases and facilitating quick development is a data dictionary. A data dictionary can function as a tool, but tools should not get in the way of the primary goal: solving the problem. Any tool used for requirements analysis must contribute to managing project complexity rather impeding progress. Capturing the data which describes the data held in the database, the metadata, must be a smooth, easy process which does not burden the momentum of discovery built in working with users. Users know what they do and therefore can describe processes far easier than data. Metadata is the essence of the means of achieving high productivity, shared databases and reduction of the inefficiencies associated with the traditional development process. [Ref. 31]

## B. METADATA

Metadata is the term applied to that data which describes other data or databases. Metadata include descriptions of the meaning of data items, the ways in which the data are used; the sources of particular data elements; the physical characteristics; and rules or restrictions on their forms or uses. Some choose to classify metadata elements into 3 types: semantic, physical characteristic, or usage information. [Ref. 32]

Semantic information is comprised of the names of the component or entities. A unique identifier should be assigned to each component even though it may be known by other names. These other names are called aliases. The documenting and tracking of aliases promotes data sharing across applications. The prevalence of aliases should not be encouraged or discouraged, but should be accepted as a fact of life. End-users call things by different names and a data dictionary, as an adjunct to the database which models the real world, should accommodate that fact.

Physical characteristics are those items or information which describe the system dependent representation of the component. The length in bytes of a data element, or the access method for a particular file are examples of physical characteristics. This category includes relationships among systems and programs which use the database. That a particular data element may be found in records x, y and z and is referred to by programs A and B is an example of physical characteristic metadata.

Usage information illuminates how and by whom data are used. This type of information enables the database administrator to trace the source of errors. The existence of usage information also permits the database administrator to assess the impact of changes on the database and helps to

develop structures to monitor and control access. If the scope of a data dictionary is widened to include usage information then it must include data about users and processes as well as data about data elements. Processes can be performed by individuals, organizations, programs or entire systems. A process can be a single transaction, the generation of a particular item such as a report, or a broad function such as production of the entire monthly payroll to include checks, vouchers and a summary report for management. In compiling usage information, responsibility for a database related task should be assigned to a particular process to enable monitoring and correction. The source of a transaction, such as entering the number of hours worked for a payroll application, should be documented and traceable. In this example, a clerk may be used to verify and enter the number of hours worked. [Ref. 33]

Metadata can also be classified as one of three types of entities. Entities are either "real world" objects or concepts such as customers, accounts, salespersons and orders. Metaentities can be classified as metadata entities, metasystem entities and meta-environment entities. [Ref. 34]

Entities which represent data objects, such as data elements, groups, records, files, databases or reports are examples of metadata entities. An element is a simple single item and is a "primitive", that is, it cannot be further divided. Elements can be accumulated, usually logically, to form other higher level metadata entities such as groups or records. A savings account number is an example of a data element. A data element can be recursive, meaning it can be formed or conceived as consisting of other data elements, yet for purposes of the particular application cannot be further subdivided. A savings account number may be separated by hyphens when printed and the

separated parts can represent different meanings, such as branch, type account, and specific customer. However, these different numbers will always be carried together as a singular item. [Ref. 35]

Two or more data elements can be collected into what is called a group. Groups can contain other groups. The elements or groups collected to form a group are usually logically related in some fashion. Thus, in the example of an address group the elements street, city, state and zip code are the logically related elements comprising the group. Groups are aggregations larger than an element but smaller than a record.

A record is a collection of one or more related data elements or groups that is treated as a logical unit. Each employee of a company, for example, would have one record and it would be a collection of these data elements and groups: name, department, employee number, age and home address. [Ref. 36]

A file is a set of occurrences of records treated as a unit. Different types of records may be contained within a single file, or it can contain many occurrences of a single type. An occurrence is where actual values have been assigned to the fields or data elements of a record, and the record therefore now models the reality of the entity on which it is based. [Ref. 37]

A database is an aggregation of DBMS files, records, groups and elements. Databases vary in both logical and physical organization depending on the data model used to design the database and the DBMS used to implement the database. [Ref. 38]

Reports are also classified as metadata entities although, unlike any of the previously discussed entities, reports are usually not stored but produced on execution of a subsystem or program. A report is a formatted



representation of information, not data. Data is manipulated and given meaning when a report is produced. Data items are facts which, without manipulation and association with other facts, have no meaning to humans.

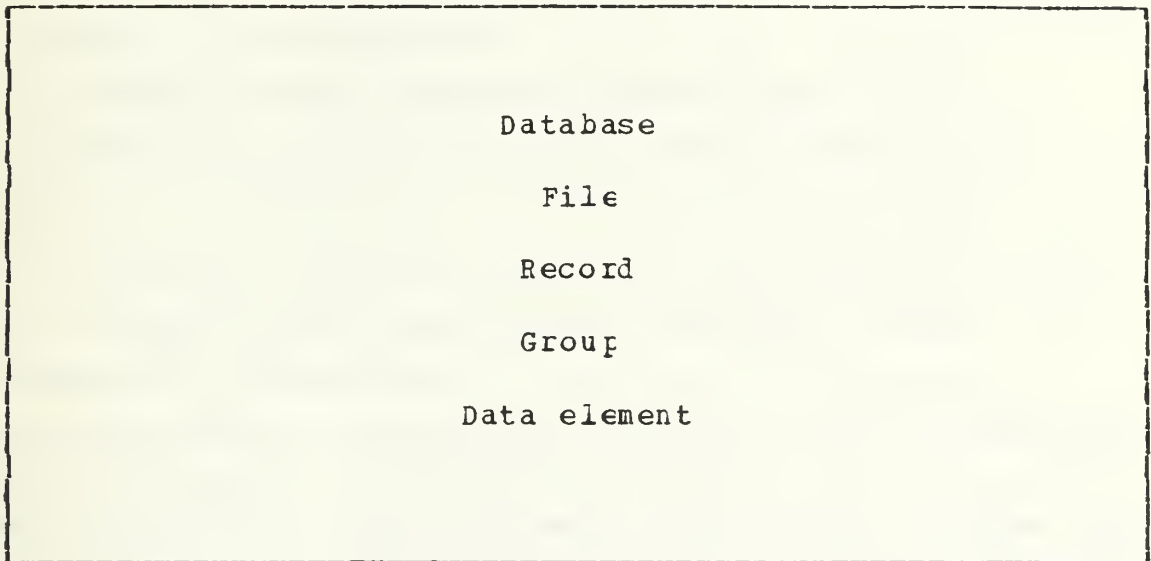


Figure 3.1 Hierarchy of Metadata Entities

Systems, programs, subroutines, and modules are all examples of metasystem entities. This class of metaentities represents processes of the data processing environment. [Ref. 39]

A system is a collection of related programs or systems which together perform a set of functions for some general purpose. For example, a payroll system might consist of many programs: one for producing checks, one for updating an employee database, and one for producing a summary report.

A program is a collection of source code statements (in assembly, compiler, query or fourth generation language) that accomplishes one or more actions. Programs contain one or more modules.



A module is a collection of executable statements which exists to accomplish one specific application function. Modules are the result of software being divided into separately addressable collections of source language statements. Modules can be categorized in one of three ways [Ref. 40] :

1. Sequential types which are referenced and executed without interruption of the application software.
2. Incremental types which can be interrupted before completion and restarted at the point of interruption.
3. Parallel types which execute simultaneously with another module in a multiprocessor environment.

A transaction is any event that requires the database information to be accessed. Transactions are characterized by type (batch or on-line), frequency (i.e. number of times per month), and use (add, delete, modify). Transactions are usually thought of as manual actions taken by a user which result in actions being taken by the software system.

Interfaces are places at which independent modules, programs, or systems meet and either act on or communicate with each other. Interfaces can be hardware, software, human or procedural components of a system. Most often there is a transfer of data across an interface which must be clearly understood. An interface description should provide a list of all data that enters and exits a component. For a single module the description should indicate what data moves across an argument list, any external world input/output information that is used and information acquired from global data items. An example of an interface description for a module which calculates the miles per gallon being used by a digital automobile dashboard system would look as follows [Ref. 41] :

Calling protocol:   CALL CALC-MPG(MPH,GPH,MPG)

where

MPH is miles per gallon (real,input)  
GPH is gallons per hour (real, input)  
MPG is miles per gallon (real, output)

No external I/O or global data are used

CALLED BY: DATA-CONV-CTL

CALLS: no subordinates

The environment constitutes another category of meta-entities. This category can be divided into many different items if it is desired that the system track the equipment configuration as well as the data. If not, then the category of physical devices is sufficiently descriptive and refers to such items as a CRT terminal, printer, disk drive, etc. [Ref. 42]

An important aspect of gathering and analyzing metadata is describing it. Descriptive adjectives, categories and properties of an entity are called attributes. Plagman and Leong-Heong cite eight generic categories of meta-entity attributes:

1. Identification attributes which designate, describe or somehow identify the entity. Entities may be known by more than one name depending on the time, place and user's frame of reference.
2. Relationship attributes denote associations between two meta-entities. For example, one entity may be used by another. Relationship attributes are sometimes similar to interfaces. A link, subroutine call, access path or mapping qualify as a relationship type attribute.
3. Representation attributes describe an entity as it is implemented. A data element is represented in terms of character (alphabetic, numeric or mixed), source, or length.

4. Statistical attributes are numerical descriptions such as frequency, estimated life, and usage. This class of attributes are useful to the Database Administrator in assessing the impact of changes and for optimizing queries.
5. Physical attributes help describe environmental and system entities. Examples are storage media, storage size, terminals and CPU model.
6. User-defined attributes are those things about a particular application such as the users, analysts, designers, customers, etc. that are helpful. A "free-form" attribute is included in this category. This is often implemented as a comment field but can be specifically named if the user desires.
7. Control attributes are descriptors for how restraint over an entity is exercised such as security classification or edit criteria.
8. Status attributes provide the life cycle status of the entity; e.g. candidate, test, operational, deleted or archived [Ref. 43].
  - a) A test status means the entity can be modified with approval of someone in project management. A test entity is not for production runs or general use.
  - b) An item categorized as operational is being used by the system on a regular basis. Modification of the item, especially in a file oriented (as opposed to a DBMS oriented) environment, may cause problems and therefore is carefully controlled.
  - c) Archived means the item is no longer used operationally anywhere in the automated components of the system. "Old timers" may still refer to archived entities and, therefore, such items should be available for reference, but not for use.

**TABLE I**  
**The Grouping of Meta-Entities**

<u>DATA</u>	<u>SYSTEM/PROCESSING</u>	<u>ENVIRONMENTAL</u>
Element	System	Physical Devices
Record	Subsystem	Computer system
Group	Program	Terminal
File	Module	Line
Database	Transaction	Users
Screen		Node
Report		Function

Indicating the status of a meta-entity is important for the purposes of project and configuration management. It is important to know whether a simple data element, a program or an entire system is either under development, being integrated or operational.

It is obvious from this discussion that the task of managing metadata can be as challenging as managing the user's data. Consequently, the same techniques can be used to manage metadata, that is, organizing it into a database, the data dictionary.

### C. DEFINITION AND OBJECTIVES OF A DATA DICTIONARY

Familiarity with what constitutes a data dictionary and the purpose for having one are requisite for later understanding of its role in requirements analysis. The need for documenting data is obvious and grows from the universally recognized need to document the two primary components of any software system, functions and data. A data dictionary is a repository of data about data and processes associated with a particular systems development effort. A dictionary often includes a glossary of terms, data characteristics, process descriptions, and some cross-referencing mechanism.

Data dictionaries come in three different forms.

1. The data documentation for a system can be made a part of either the requirements specification or a design document.
2. A data dictionary can exist as a separate printed document.
3. A data dictionary can be an automated data base.

Data dictionaries can be expanded to include information about the physical location of the data. In that case it becomes a data dictionary/directory. If either the data dictionary or data dictionary/directory uses software to store and access the data, it is considered a system. The following delineates feasible manifestations of a dictionary:

1. A data dictionary is a repository for data about data and functions.
2. A data dictionary system is a software system to store and access data about data and functions.
3. A data dictionary/directory is a repository about both logical and physical information for data and functions.



4. A data dictionary/directory system is a software system to store and access both logical and physical information about data and functions.

Several criteria are worth examining in deciding which way to document data. These are:

1. Estimated size and complexity of the system.
2. Extent of sharing of data between systems.
3. Use of a generalized DBMS.
4. Availability of automated data dictionary software.
5. Number of data descriptions.
6. Complexity of data structures.
7. Volatility of data descriptions.
8. Number of analysts and programmers involved.
9. Number of organizations involved.
10. Amount of clerical assistance available.
11. Uses of the data documentation.

The greater the number and strength of these factors, the earlier a decision must be made about which means to document data, and the stronger the probability that that means will be an automated system. Regardless of form a data dictionary should document several different components.

All files, record types, items within records and pertinent information about the use of the data should be included in the data dictionary. Pertinent information includes such items as physical description, logical description, and functional description of how the data is used. Further, it elaborates on how the data is changed, how it is removed, and how integrity is ensured.

Based on the information the dictionary holds, it may satisfy many objectives [Ref. 44]. Generally any data dictionary should provide facilities and formats for documenting the information collected during all stages; analysis, design and implementation. Further, it should

help the analysts follow any of the structured methodologies available for system development. Data descriptions, not only processes, should be traceable back through program functions to the original statement of requirements. The dictionary should be readily extensible to accommodate growth or expanded application. Its existence should help achieve the establishment of a glossary of terms and thus provide for standard terminology. Analysis can be accomplished using the proper terms in the proper context in dealing with the user's particular application area. Semantics are important for both understanding the user's area and for tailoring the system to his frame of reference. Modules which are available to all systems should be identified via a generic cross-reference capability.

Going beyond the analysis phase, the dictionary can provide assistance in generating manageable and complete test data. Problems which arise associated with aliases and acronyms can be resolved via a data dictionary. As an adjunct to configuration management, a dictionary provides centralized control for system changes. Again going beyond the analysis phase, a dictionary can serve as a reference and guide in training users and for design evolution. Maintenance efforts can be assisted in a similar vein in that programmers unfamiliar with specifics can refer to the dictionary as a guide.

Having met a majority of these objectives, a dictionary should be capable of answering several questions [Ref. 45], [Ref. 46].

1. What kind of validity tests have been applied to this data type?
2. Who is authorized to update this data item?
3. What modules, programs or systems use this data type?
4. What are the valid ranges of values for this data?  
(domain)

5. What security level is applied to this item?
6. Who is allowed access to the data?
7. By what other names is the data type known in various application environments? (the differences should be minimized)
8. In what reports does this data type appear?
9. What is the input source for this data type?
10. What are all the data items used by a given program or report?

Data dictionaries may have problems, particularly if they are not well organized. If a dictionary is fully used, meaning that if it is an active dictionary which serves a controlling function, then it becomes a centralized controlling facility. Centralized control of data is not always desirable or appropriate. For a data dictionary to get to this point, considerable effort must be expended. Therein lies another problem in that development and implementation, to be useful, requires careful thought and considerable effort [Ref. 47]. Another problem is the lack of a methodology which ties the use of a dictionary into the overall system development effort. Dictionaries need to be able to support and measure the documentation progress. Unfortunately the means of describing primitive entities in different commercial data dictionary packages is dissimilar. Few dictionaries have specific provisions for supporting a structured analysis technique [Ref. 48]. Features which are important for database design, but absent from many current products are support for distinguishing between a data element and its domain and the lack of a graphic representation capability. In response to these problems this thesis proposes that a model that is easily understood, the relational, be used to organize a data dictionary.

The relational model is appropriate for prototyping with data dictionaries for several reasons. As discussed earlier, the ease of use, availability of query language and data independence make relational-based DBMS good tools for prototyping. The relational model is good for data dictionary implementation because many relational DBMS have a "triggering" mechanism causing a program to be invoked on some data event or condition. This is an essential feature for a data dictionary to be active, that is, to serve as a monitor of database activity to ensure data integrity. An additional advantage is the schemas of relational systems are tables. This can reduce the work required to implement the dictionary system. For these reasons, and because relational DBMS are now available for microcomputers, a prototype data dictionary has been developed and is described in the next chapter.

#### IV. A DATA DICTIONARY BASED ON THE RELATIONAL MODEL

##### A. INTRODUCTION

The office of the Deputy Chief of Staff for Plans (DCSPLANS), the US Army Military Personnel Center (MILPERCEN), has the responsibility to develop and maintain computer based modelling capabilities to support personnel policy development for the Army. DCSPLANS has subordinate branches each of which uses and/or maintains models. In a prototype effort to assist DCSPLANS organize and account for these information resources, a relational dictionary system was developed using the dBase II relational DBMS.

The prototype system is capable of running on an IBM Personal Computer. A relational based DBMS was used because of the features previously discussed. A microcomputer-based version was used to facilitate transfer of the prototype from the development site at the Naval Postgraduate School to MILPERCEN at Alexandria, VA.

The dictionary accounts for several different relations including reports, offices, models, programs, files, elements (which are partitions of files, rather than data elements) and fields. The generalized format of these relations is as follows:

REPORT( rname, full\_name, number, status, sec\_class,  
output\_format, purpose, description, perdiodicity)

OFFICE( ofc, oname, bldg, room, phone, comments, poc)

MODEL( mdl, model\_name, methodology, status, location,  
purpose, description, runame, perdiodicity)



PROGRAM( pname, version, location, language, purpose,  
description, status, peridodicity)

FILE( fname, full\_name, last\_update, media, status,  
sec\_class, purpose, description, poc)

ELEMENT( elname, version, date, seq\_num, size, poc, purpose,  
location)

FIELD( fld, length, data\_type, null\_status, source, domain,  
description, purpose, status, sec\_class, poc)

For a detailed explanation of the attributes for these  
relations, see Appendix B, Data Dictionary Data Elements.

Relationships amongst the various relations are tracked  
by having relations with a verb name reflecting how one  
entity relates to another. For example, since a MODEL may  
be used by many different OFFICES, a USES relation is  
included in the dictionary. Its format is as follows:

USES( ofc, ename, etype). An example occurrence of this  
relation would be:

USES(DAPC-PLF, P3M, MODEL)

which states that the office DAPC-PLF uses the P3M model.

There are six of these verb type relations. In addition to  
the USES relation these are CONTAINS, MAINTAINS, OUTPUTS,  
INPUTS, and CALLS. The generalized format of each of these  
is as follows:

CONTAINS( ename1, etype1, ename, etype)

MAINTAINS( ofc, mdl, model\_poc)

OUTPUTS( ename1, etype1, ename, etype)

INPUTS( ename1, etype1, ename, etype)

CALLS( ename1, etype1, ename, etype)

The attribute ename stands for entity name and the attribute entity type stands for entity type. In general, these attributes can be instantiated by any entity (e.g. model, program, file, element, etc.) The specific relationships these relations track (see Figure 4.1) are as follows:

1. An OFFICE uses many different REPORTS.
2. An OFFICE uses or maintains many MODELS.
3. A MODEL is used or maintained by many different OFFICES.
4. A MODEL calls many different PROGRAMS.
5. A MODEL outputs many different FILES.
6. A MODEL inputs many different FILES, i.e. requires many different FILES as inputs.
7. A MODEL outputs many different REPORTS.
8. A FILE contains many different ELEMENTS.
9. An ELEMENT contains many different FIELDS.
10. A FIELD is contained by many different ELEMENTS.

There are two special relations which are neither pure entities nor verb descriptors. These are ALIAS and CATEGORY.

The ALIAS relation provides a means of tracking entities by all known names. Often an entity is given one or more names other than that by which it's officially known. For the users the other name, the alias, is the natural way of referring to the entity and therefore the dictionary should accomodate that. The Personnel Policy Projection Model being called "Pimmy" is an example of an alias.

The CATEGORY relation provides the basis for Key Word In Context (KWIC) capability. Further, it allows the user to categorize different types of entities and relate them in an ad-hoc fashion. For example, if several files and programs were the responsibility of a particular programmer to audit, all those different entities could be entered as category records with that programmer's name as the particular category.

The format of the ALIAS and CATEGORY relations is as follows:

ALIAS( ename, etype, aname)

CATEGORY( ename, etype, cat)

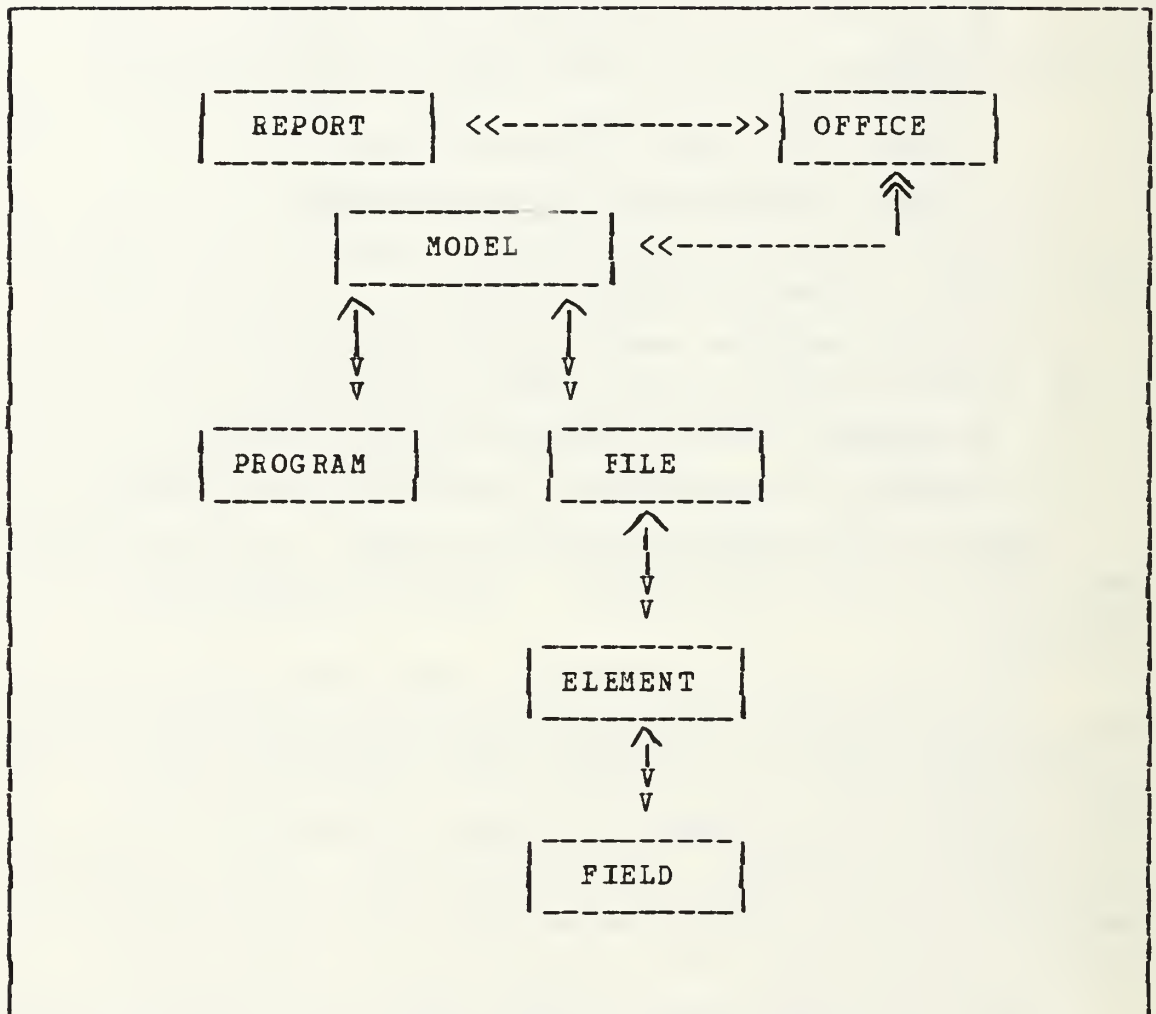


Figure 4.1 Bachman Diagram of Entities

Structuring the metadata as shown above offers advantages for ease of learning and using the data dictionary. The logical structure is based on a

commercially successful data dictionary system, DataManager. The data dictionary is a database and therefore models the real world in a limited fashion. By taking a data-centered viewpoint wherein all entities are categorized as either objects (i.e. models or reports), or processes (i.e. uses, contains, maintains) which act on objects, an easy-to-understand model emerges. Consequently, use of the dictionary is facilitated so that it becomes a tool, rather than a burden to the data system developer/maintainer/user. This will be substantiated by the following discussion of the system structure and query/report capabilities.

## **B. SYSTEM STRUCTURE AND USER INTERFACE**

The software of the system was structured to provide all the possible functions one might wish in a database system. These are the ability to add, change, or delete records and to provide reports and queries (see Figure 4.2). Because the system is a prototype and specific user needs are unknown, all functions are provided. Depending on user feedback and observations made by the analyst, these functions can be expanded to offer different dialogues or be converted to batch procedures.

The user interface for any interactive system is an important criterion for success. Because a data dictionary system should be a tool to facilitate software system development and maintenance, it should provide a friendly user interface. This data dictionary system is menu and prompt driven. The software prompts for all necessary data and actions to develop and maintain intersection records (in this system called processes or verbal relations such as CONTAINS or USES). Consequently the user manual for the system is not a long and trying document to use.

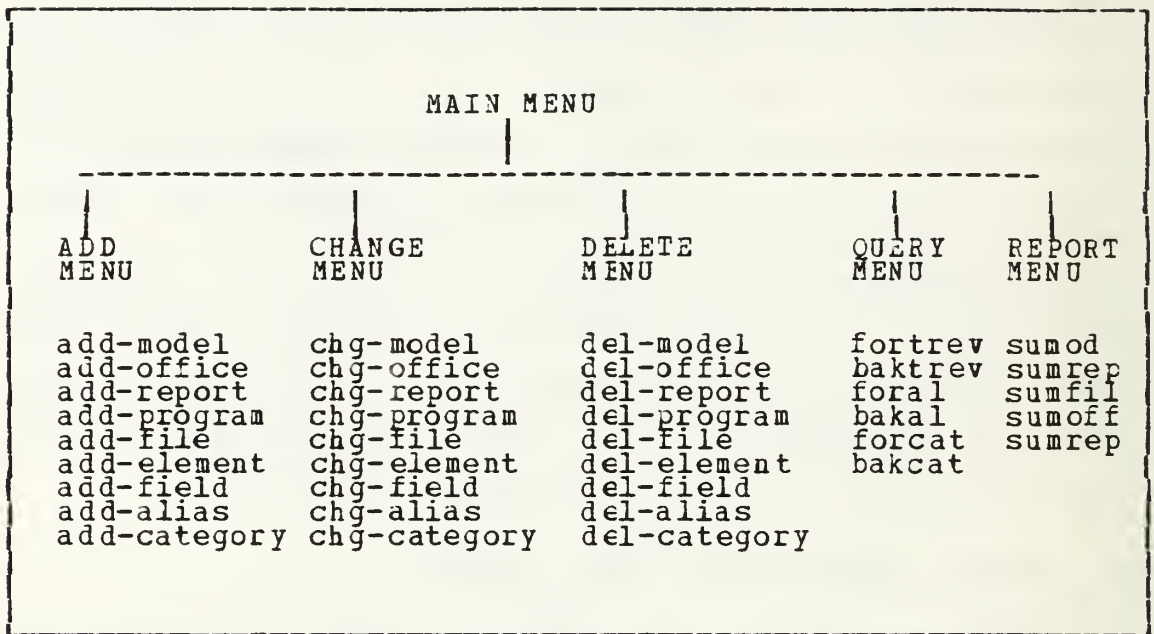


Figure 4.2 Hierarchical Chart of Software

The query interface is particularly friendly. The system, named dDICT II, provides two types of queries, forward and backward. Forward means the user is presented columns of relations from which to pick. The columns, following the forward flow of the English language, are subject, verb and objects. The user provides the unique name of an entity beside its type in the subject column; marks 1 appropriate verb column and 1 appropriate object column. For example, to answer the question "What models does the office with symbol DAPC-PLF use ?" the Figure 4.3 depicts how the screen of the FCRTREV program would appear. The Figure 4.4 depicts how output from this query would appear. Verb and object choices must be appropriate for the subject in that it is not possible to get any response from a query marked for which there is no relationship defined. For example, marking the screen of the FORTREV program to answer the question "What models contain what fields ?" would elicit no response when processed.



<u>SUBJECTS</u>		<u>VERBS</u>		<u>OBJECTS</u>
Office:DAPC-PLF	:	Uses:X:		Reports: :
		Maintains: :		Models:X:
Model:	:	Calls: :		Programs: :
		Inputs: :		Files: :
		Outputs: :		
File:	:			Elements: :
Element:	:	Contains: :		Fields: :

To quit put an X in this field : :

Put the unique name of the entity you are querying about in the field next to the entity's type. Then place an X in one appropriate verb field for that entity type and an X in 1 appropriate object field.  
 I.E. Office:DAPC-PLF : Uses:X: Models:X:  
 To process query move cursor through remaining fields using RETURN key.

Figure 4.3 Example of a Forward Retrieval

The office DAPC-PLF uses the following models  
 P3M  
 OAS  
 Strike any key to continue...  
 WAITING

Figure 4.4 Sample Output From a Forward Retrieval

The backwards approach is implemented by presenting from left to right columns of objects, verbs and then subjects. Thus, to answer a question such as "What offices

use the P3M model ?" see Figure 4.5 for how the the screen of the BAKTREV program would be marked. This query will produce the unique names of the offices that use the P3M Model and Figure 4.6 shows how the output would appear.

<u>OBJECTS</u>	<u>VERBS</u>	<u>SUBJECTS</u>
Model:P3M	: Is used by:X:	Office:X:
Report:	: Is maintained by: :	
Program:	: Is called by: :	Model: :
File:	: Is inputted by: :	
	: Is outputted by: :	
Element:	: Is contained in: :	File: :
Field: :		Element: :
To quit put an X in this field: :		
Enter the name of the entity besides its type and then mark 1 verb field and 1 subject field. To process query move cursor down through remaining fields using RETURN key.		

Figure 4.5 Example of a Backward Retrieval

Because the ALIAS and CATEGORY relations are each a special case, they are given their own respective forward (FORAL & FORCAT) and backward (BAKAL & BAKCAT) query programs. For example, if you wish to know all the aliases that exist for all models in dDICT II you would execute the FORAL program and mark the menu as in Figure 4.7 and Figure 4.8 shows how the output would appear.

If you wish to know all aliases for an entity for which you knew the name you would use the BAKAL query. For example, if you wanted to know all the aliases for the P3M

The model P3M is used by the following offices

DAPC-PLP  
DAPC-PLT  
DAPC-PLS  
DAPC-PLF

Strike any key to continue ...  
WAITING

Figure 4.6 Output From a Backwards Retrieval

Model:X:	File: :
Program: :	Element: :
Report: :	Field: :

Enter an X in the field beside the type of entity  
for which you want all names and aliases. Mark only 1  
field.

Figure 4.7 Example of FORAL Menu

model you would mark the menu of the BAKAL query as in Figure 4.9 and Figure 4.10 shows how the output from that query would appear.

The FORCAT program permits you to determine all occurrences of a type of entity and in what category each occurrence falls. For example, if you wish to know what models had been categorized in some way see Figure 4.11 for how to mark the menu of the FORCAT program. Figure 4.12 depicts how the output would appear. See Figures 4.13 and 4.14 regarding use of the BAKCAT program.

The following models have the associated aliases

P3M	PIMMY
P3M	PAIN
OAS	ASSETER

Strike any key to continue ...  
WAITING

Figure 4.8 Example of Output From FORAL Query

Model:P3M	:	File:	:
Program:	:	Element:	:
Report:	:	Field:	:

Enter the name of the entity in the field  
beside its type. Provide only 1 name.

Figure 4.9 Example of BAKAL Program

The model P3M has the following aliases

PIMMY  
PAIN

Strike any key to continue ...  
WAITING

Figure 4.10 Example of Output from BAKAL

```
Model:X:                File: :  
Program: :              Element: :  
Report: :              Field: :  
Enter an X in the field besides the type of  
entity for which you want all names and associated  
category. Mark only 1 field.
```

Figure 4.11 Example of Menu for FORCAT Program

```
The following models are in the associated category  
OAS                OFFICER  
Strike any key to continue ...  
WAITING
```

Figure 4.12 Example of Output From FORCAT

For more extensive information about a particular Model, File, Office, or Report, the Report subsystem exists. This subsystem has a menu from which the user selects the type of report program to execute. The system will then prompt the user for the unique name of an entity. From that the system will then provide a printed summary report which provides all entities which have a direct relationship with the entity named. Thus, if the user provides the unique name of a Model, the printed report would list:

1. All offices which use the Model.



```

Model:OAS      :           File:           :
Program:       :      Element:           :
Report:        :           Field:           :
Enter the name of the entity in the field beside its
type. Provide only 1 name.

```

Figure 4.13 Example of Menu for BAKCAT Program

```

The model OAS   is in the associated category
OFFICER
Strike any key to continue
WAITING

```

Figure 4.14 Example of Output for BAKCAT Program

2. All offices which maintain the Model.
3. All programs the Model calls.
4. All reports the Model outputs.
5. All files the Model either inputs from or outputs to.

For example, if you wished to have a summary report for the P3M model you would select option 1 from the Reports menu for execution of the SUMOD program. The system would then prompt you as follows:

```

Please enter the unique name of the Model
Hit the space bar and RETURN to quit:P3M

```

Figure 4.15 shows how the output would appear.

The Model P3M

Is used by the following offices:

DAPC-PLP  
DAPC-PLT  
DAPC-PLS  
DAPC-PLF

Is maintained by the following offices with  
the point of contact:

DAPC-PLF KALINICH

Inputs the following files:

P3MDATA

Outputs to the following files:

P3MOUTPUT1

Outputs the following reports:

EPMR  
P3MREP1

Calls the following programs

P3MPROGUNO1  
P3MPROGDOS1  
P3MPRCGTRES1

Strike any key to continue  
WAITING

Figure 4.15 Example of Report Output

Providing these query and report capabilities makes the dictionary more than a repository but a fully capable tool. Use of the dictionary to manage data is not merely made possible but truly facilitated. The data administrator can quickly input or retrieve metadata as appropriate to impose order, control and effective management of a critical part of the organization's resources. Implementation and use of a dictionary marks an important step in an organization towards knowing what data assets it has and most

importantly, what data assets it needs. Needs are requirements the specification of which must be precise to be implementable. For example, knowing exactly what fields occur in what elements and, therefore, what else we need to accomplish some function is a critical question to be answered in defining requirements. Having an automated and interactive means of answering such questions provides a decided advantage over using textual documents. This dictionary thus provides an important tool from which the data administrator can more expeditiously manage the organization's current data processing capabilities.

Because the user of this dictionary is located across the country from the development site, a demonstration version of the dictionary was created and populated with example data (see Figures 4.16 and 4.17). The demonstration version can be used to learn the capabilities of the system without fear of destroying anything. Once confident with the operation of the demonstration version, the user can migrate to the production version.

Because dDICT II is a prototype it is missing features a refined and full production system would have. Most importantly, the software doesn't enforce relational constraints. Thus, some entities can be present without others also being present. For example, a program can exist without there also existing a model which calls that program. It remains to be determined how important it is for metadata integrity to have the software enforce relational constraints. Further, dDICT II does not enforce what can be entered for a particular attribute of an entity. Because many of the attributes contain narrative text it is not possible to develop precise logic for what can and cannot be entered. However, this is a refinement which should be added once the user has migrated to the production version and populated it with real metadata. The attributes

Model: OAS

Reports: OPMR, OASREP1

Users: DAPC-OPD, DAPC-PLF

Maintainers: DAPC-OPD

Programs: OASPRGUN01 , OASPRGDOS1

Input file: OPDOASDATA

Output file: OASOUT

Element: INSPEC

Element: ADSPEC

Fields: grade  
inspecode  
commsvr  
degrees  
dor

Fields: name  
grade  
adspecode  
yrssvc

Element: URATE85

Element: CPMRECAP

Fields: oct  
nov  
dec  
jan  
feb  
mar  
apr  
may  
jun  
jul  
aug  
sep

Fields: oct  
nov  
dec  
jan  
feb  
mar  
apr  
may  
jun  
jul  
aug  
sep

Aliases: OAS ASSETER  
URATE85 UTEY

Categories: URATE85 numeric  
OAS officer  
OASPRGUN01 cobol  
OASPRGDOS1 development1

Figure 4.16 First Example of Demonstration Data

used in describing metadata entities lack precision and need tightening in follow-on efforts. Precise domains need be

Model: P3M

Reports: EPMR, P3MREP1

Users: DAPC-PLP, DAPC-PLT, DAPC-PLS

Maintainers: DAPC-PLF

Programs: P3MPROGUNO1, P3MPROGDOS1, P3MPROGTRES1

Input file: P3MDATA

Output file: P3MOUTPUT1

Element: SEEPAGE84

Element: TNGPROG84

Fields: mosto  
      mosf  
      e1 through e8

Fields: sl  
      codes  
      npstng  
      pstng

Element: VALIDMOS

Element: CRSLENATR84

Fields: cmf  
      etsyr  
      etsmd  
      tdatyr  
      tdatmn  
      bygrade  
      egrade  
      title

Fields: wks  
      att

Element: AUTHCUR

Fields: e3 through e9

Aliases:	P3M	FIMMY
	P3M	FAIN
	EPMR	IPPER
	P3MPROGUNO1	FIMMY1
	P3MPROGDOS1	FIMMY2

Categories: P3MPROGUNO1 cobol  
            P3MPROGDOS1 fortran

Figure 4.17 Second Example of Demonstration Data

determined, documented and enforced by edit criteria.  
Refining the user's specification of metadata will



facilitate more extensive query and report capabilities. It is only possible with more specific domains that a more user friendly query interface can be developed, possibly using artificial intelligence techniques. Using artificial intelligence techniques will lead to a more useful tool capable of assessing the impact of change on the databases through sensitivity analysis on the dictionary. Unfortunately, the prototype version is not "instrumented" to capture user peculiarities for enhancing friendliness and performance. Because dBASE II was used, the system suffers poor performance in comparison to what would be enjoyed using dBASE III. A full production version would be inordinately slow if not implemented in dBASE III.

## **V. CONCLUSIONS**

This thesis has investigated the use of prototypes and data dictionaries as vehicles for assisting requirements analysis. It has explained the advantages and disadvantages of prototypes; provided guidance for their use; explained tools that are needed and provided a data driven approach to prototyping. It has shown that prototyping is a useful adjunct to the systems development life cycle and is most appropriately applied to Management Information System applications. The most significant contribution of prototyping is the establishment and maintenance of effective communications with the user during system development.

This thesis has shown the importance of having a data dictionary mechanism to capture the output of prototyping. Further, it has explained metadata and its importance in building a system and understanding both the user's requirements and project management's requirements. Clearly metadata, like user data, must be organized into a database, a data dictionary system. Organization of a data dictionary can be in accordance with one of many different data models, but the relational offers the most promise for future use. Consequently, the thesis includes a prototype relational dictionary system.

The dictionary system demonstrates the feasibility of using:

1. the relational model for a data dictionary.
2. micro-computers in a prototype effort.

Although of limited functionality, the prototype was quickly developed and offers considerable flexibility in its structure to serve as an on-going aid in developing the data

administration function in the user's organization. DDICT II is living proof of the viability of prototyping with data dictionaries as a means of organizing and understanding the data and user's needs in system development.

## APPENDIX A

### USER MANUAL

#### A. INTRODUCTION

This manual explains only what is necessary to get started using the system. The system is self-prompting and, therefore, an extended narrative is unnecessary. This manual explains the use of both the demonstration system and the actual dictionary. The demonstration system resides on floppy disks and requires a different initialization procedure. If you are using the demonstration system, proceed to the next section. If you want to use the actual installed dictionary, skip to section C. After initialization both systems operate identically so, upon completing initialization, proceed to section D, MAIN menu.

#### B. INITIALIZING THE DEMONSTRATION SYSTEM

1. If the system is on, turn it off and remove all floppy disks.
2. Insert the floppy disk marked 'dBASE II' into drive A and the floppy marked 'DEMONSTRATION' into drive B.
3. Turn on the system (all components: computer, monitor and printer).
4. Upon being prompted for the date, you may enter it in the general form: MM-DD-YY , or simply strike the RETURN key. The same is true for the following prompt for the time. If you wish to have the current time available to the system, the general form of your reply is: HH:MM .
5. Upon correctly answering the time and date prompts you will receive the following on the screen:

A>

6. Now type in 'DBASE' and hit the RETURN key.
7. You will hear the 'A' disk drive whirr and its light come on. After a few seconds a message will appear:

Copyright (C) 1982 RSP Inc. \*\*\* dBASE II/86 Ver 2.4  
1 July 1983

.\_

The blinking underline character following the period is the cursor. The period is the prompt used by dBASE II and indicates that you have accessed the dBASE II software.

8. You should now type the following in response to the prompt:

SET DEFAULT TO B:

9. Upon receiving the period prompt you are now ready to use the demonstration version of the data dictionary. At this point proceed to section D, MAIN menu.

### C. INITIALIZING THE INSTALLED DICTIONARY

This manual assumes that the user is familiar with the IBM PC sufficiently to start it up and move to the appropriate working directory upon which the dictionary system is installed. For the purpose of this manual it assumes that directory (disk drive) is C. Once you have moved to that disk you are ready to start up the dbase II software. At this point type in "dbase" and then hit the RETURN key. You will hear the "C" disk drive whirr and its light come on. After a few seconds a message that reads as follows will appear:



.-

The blinking underline character following the period is the cursor. The period is the prompt for dBASE II and indicates that you have the accessed dBASE II software. Upon receiving the period prompt you are now ready to use the actual version of the data dictionary. Proceed to the next section, MAIN menu.

#### D. MAIN MENU

Type in the following and the main menu for the data dictionary will appear: DO MAIN (all capital letters aren't necessary, but it's best for this particular system to set the "caps lock" key so that you are always in uppercase. To set the "caps lock" key just strike the key so marked once).

The MAIN menu permits selection of the major function you wish to perform. Upon selecting a major function, enter the number associated and another menu will appear. Please wait for the "WAITING" message to appear before typing the number of the major function you want. Upon selecting a major function another menu will appear (See example of the Main Menu on the next page). Use the instructions of this manual for working with those menus. After a period of use you will become familiar with the specific functions of the system, such as adding a model record. If you find the use of the menus tedious at that time, you can execute any specific program by entering "DO program name". For example, to directly execute the program to change an office record you would enter "DO CHG-OFFICE". To help you learn the specific program names they are provided in the menus in parentheses beside the explanation of a choice. This means that once you have the period prompt mentioned above, you

can directly accomplish a particular function by executing the associated program and not stepping through the menus.

This is the MAIN menu for the data dictionary system.

The system allows you to add, change, or delete records and to process queries. Please enter the number of the function you wish to perform:

0 to quit

1 add a new record	(ADD-MENU)
2 change a record	(CHG-MENU)
3 delete a record	(DEL-MENU)
4 run a query	(QERY-MENU)
5 print a report	(REPT-MENU)

Enter the selection here

WAITING

#### E. ADD SUBSYSTEM

There are two methods of accessing the ADD-MENU; either by typing in the command 'DO ADD-MENU' or by selecting option #1 from the MAIN-MENU. In either case, the following menu will appear:

##### ADD MENU

0 to quit

1 Model (ADD-MODEL)	5 File (ADD-FI)
2 Office (ADD-OFFICE)	6 Element (ADD-EL)
3 Report (ADD-REP)	7 Field (ADD-FE)
4 Program (ADD-PRO)	8 Alias (ADD-AL)
	9 Category (ADD-CAT)

Enter the selection here

WAITING 0

Selecting 0 from the ADD-MENU will return you to either the MAIN-MENU or a blank screen (if the 'DO ADD-MENU' command was used).

To add a new record, press the number associated with the entity's type. The system will prompt you for the unique name of the entity you wish to add. If that name is not truly unique, the system will tell you so and not permit you to add a new record. If the name is unique a screen will appear with the fields highlighted to include blank spaces for entering data for the new record. Simply fill in the appropriate data items and move from field to field (USING THE CURSOR) until you have filled in all the blank fields for which you have data. Once you have assured yourself that all data items you have entered are correct, hit the RETURN key and the record will be added to the database. (Note: after the RETURN key has been hit changes can only be made using the CHANGE subsystem)

Once you have entered a record by hitting the RETURN key you will be prompted if you wish to add another record to the database. To stop adding records, hit the space bar and RETURN. As soon as the RETURN key is hit you will be returned to either the ADD-MENU or the dbase II system.

#### F. CHANGE SUBSYSTEM

There are two methods of accessing the CHANGE menu , either typing in the command "DC CHG-MENU" and striking the return key or selecting option 2 from MAIN menu. In either case the CHG menu will appear as follows:

This is the CHANGE menu. It permits you to change records in any file. Please enter the number of the function you wish to perform.

0 to quit

1 Model (CHG-MODEL)

5 File (CHG-FI)

2 Office (CHG-OFFICE)	6 Element (CHG-EL)
3 Report (CHG-REP)	7 Field (CHG-FE)
4 Program (CHG-PRO)	8 Alias (CHG-AL)
	9 Category (CHG-CAT)

Enter the selection here

WAITING

When selecting from the menu always wait for the system to prompt you with the words "WAITING". Selecting 0 will return you to either the MAIN menu or a blank screen (if the "DO CHG-MENU" command was used).

Regardless of whatever type entity you wish to change, the system will require you to provide the entity's unique name. To quit changing records, hit the space bar and RETURN. The system will then find the record for that entity and present it on the screen for editing. Having completed the editing, the record can be stored back in the dictionary by moving the cursor down through the last field using the RETURN key. If there are any relationships for which the entity is an object of, the particular program you are using will prompt you if you wish to add or delete entities associated with the one you just changed. For example, if you edited the record for a particular FIELD, then the system will prompt you if you wish to add or delete verbal type records for ELEMENTS that contain the FIELD. A change in the unique name of an entity will be automatically changed in all verbal type records for you by the software. For example, if you changed the unique name of a FIELD the software will change all CONTAINS records accordingly for you.

## G. DELETE SUBSYSTEM

There are two methods of accessing the DELETE menu , either typing in the command "DO DEL-MENU" and striking the return key or selecting option 3 from MAIN menu. In either case the DELETE menu will appear as follows:

This is the DELETE menu. It permits you to delete records from any file. Please enter the number of the function you wish to perform.

0 to quit

- |                       |                      |
|-----------------------|----------------------|
| 1 Model (DEL-MODEL)   | 5 File (DEL-FI)      |
| 2 Office (DEL-OFFICE) | 6 Element (DEL-EL)   |
| 3 Report (DEL-REP)    | 7 Field (DEL-FE)     |
| 4 Program (DEL-PRO)   | 8 Alias (DEL-AL)     |
|                       | 9 Category (DEL-CAT) |

Enter the selection here

WAITING

Select the type entity you wish to delete by entering the number associated with the entity's type. The particular program selected will prompt you for the unique name of the entity. To quit deleting records, hit the space bar and RETURN, otherwise type in the name and hit RETURN. If you have given the unique name correctly, the system will find that entity's record and ask if you are sure that is the one you wish to delete. If you answer Y, the system will delete the associated record, not merely mark it for deletion. Further, the system will automatically delete all associated verbal type records for the particular entity deleted. For example, if you chose to delete the FIELD named Grade, and that field is contained in the ELEMENT Aspec, then all the CONTAIN records showing Aspec as containing the field will also be deleted.



## H. QUERY SUBSYSTEM

There are two methods of accessing the QUERY menu, either by typing in the command "DO REPT-MENU" and striking the return key or selecting option 4 from the MAIN menu. In either case the QUERY menu will appear as follows:

This is the QUERY menu. It permits you to make ad-hoc inquiries. Queries are either of the 'forward' type where an entity is the subject and you specify the verb and object, or of the 'backward' type where the entity you give is the object of an action and subject.

### QUERY MENU

- 1) Forward type (i.e. P3M calls what programs ?) (FORTREV)
- 2) Backward type (i.e. P3M is used by what offices ?) (BAKTREV)
- 3) Forward type for aliases ( indicate type and system provides all aliases) (FORAL)
- 4) Backward type for aliases (give entity name and system (BAKAL) provides all aliases for that particular entity)
- 5) Forward type for categories (indicate type and system gives categories) (FORCAT)
- 6) Backward type for categories (give entity name and (BAKCAT) system gives all categories that entity is in)

0 to quit

Enter the selection here

WAITING 0

The system provides two types of queries, forward and backward. Forward means the user is presented columns of relations from which to pick. The columns, following the forward flow of the English language, are subject, verb and

object respectively. The user provides the unique name of the entity beside its type in the subject column; then marks one appropriate entry in the verb column and one appropriate entry in the object column. For example, to answer the question "What models does the office with symbol DAPC-PLF use ?" see Figure A.1 which depicts how the screen of the FORTREV program would appear and Figure A.2 depicts the output. Verb and object choices must be appropriate for the subject in that it is not possible to get any response from a query marked for which there is no relationship defined. For example, marking the screen of the FORTREV program to answer the question "What models contain what fields ? " would elicit no response when processed.

SUBJECTS	VERBS	OBJECTS
-----	-----	-----
Office:DAPC-PLF :	Uses:X:	Reports: :
	Maintains: :	Models:X:
Model: :	Calls: :	Programs: :
	Inputs: :	Files: :
	Outputs: :	
File: :		Elements: :
	Contains: :	
Element: :		Fields: :
To quit put an X in this field : :		
Put the unique name of the entity you are querying about in the field next to the entity's type. Then place an X in one appropriate verb field for that entity type and an X in 1 appropriate object field.		
I.E. Office:DAPC-PLF :	Uses:X:	Models:X:
To process query move cursor through remaining fields using RETURN key.		

Figure A.1 Example of a Forward Retrieval

The office DAPC-PLF uses the following models

P3M  
OAS

Strike any key to continue...  
WAITING

Figure A.2 Example of Forward Query Output

The backwards approach is implemented by presenting, from left to right, columns of objects, verbs and then subjects. Thus, to answer a question such as "What offices use the P3M model ?" the screen of the BAKTREV program would be marked as shown in Figure A.3 and Figure A.4 depicts how the output from this query would appear. Because the ALIAS and CATEGORY relations are each a special case, they are given their own respective forward (FORAL & FORCAT) and backward (BAKAL & BAKCAT) query programs.

## I. REPORTS SUBSYSTEM

There are two methods of accessing the REPORT menu, either by typing in the command "DO REPT-MENU" and striking the RETURN key or selecting option 5 from the MAIN menu. In either case the menu will appear as follows:

<u>OBJECTS</u>	<u>VERBS</u>	<u>SUBJECTS</u>
Model:P3M	: Is used by:X:	Office:X:
Report:	: Is maintained by: :	
Program:	: Is called by: :	Model: :
File:	: Is inputted by: :	
	: Is outputted by: :	
Element:	: Is contained in: :	File: :
Field: :		Element: :
To quit put an X in this field: :		
Enter the name of the entity besides its type and then mark 1 verb field and 1 subject field. To process query move cursor down through remaining fields using RETURN key.		

Figure A.3 Example of a Backward Retrieval

The model P3M	is used by the following offices
DAPC-PLP	
DAPC-PLT	
DAPC-PLS	
DAPC-PLF	
Strike any key to continue ...	
WAITING	

Figure A.4 Output From a Backwards Query

This is the REPORT menu. It permits you to print summary reports from the dictionary.

#### REPORT MENU

- 1) for a specific Model all users, maintainers, input & output files, programs called (SUMOD)
  - 2) for a specific Report all users and models outputting it (SUMREP)
  - 3) for a specific Office all models used, all models maintained, and reports used. (SUMOFF)
  - 4) for a specific File all models it provides input to, receives output from, and elements it contains (SUMFIL)
- 0 to quit

Enter the selection here

WAITING

Before executing any Report menu option be sure the printer is both turned on and is "on line". Failure to have the printer turned on may result in your being thrown out of the data dictionary system and dBASE II and, therefore, to reinitialize. Select the report program you wish to execute. The system will then prompt you for the unique name of the entity concerned. For that specific entity, the system will then provide a printed summary report which provides all entities which have a direct relationship with the entity. Thus, if the you provide the unique name of a Model, the printed report would list:

1. All offices which use the Model.
2. All offices which maintain the Model.
3. All programs the Model calls.



4. All reports the Model outputs.

5. All files the Model either inputs from or outputs to.  
For example, if you wished to have a summary report about  
for the P3M model you would select option 1 from the Reports  
menu for execution of the SUMOD program. The system would  
then prompt you as follows:

Please enter the unique name of the Model  
Hit the space bar and RETURN to quit:P3M

The Model P3M

Is used by the following offices:

DAPC-PLP  
DAPC-PLT  
DAPC-PLS  
DAPC-PLF

Is maintained by the following offices with  
the point of contact:

DAPC-PIF KALINICH

Inputs the following files:

P3MDATA

Outputs to the following files:

P3MOUTPUT1

Outputs the following reports:

EPMR  
P3MREP1

Calls the following programs

P3MPROGUNO1  
P3MPROGDOS1  
P3MPROGTRES1

Strike any key to continue  
WAITING

Figure A.5 Example of Report Output

## J. TERMINATING A SESSION

Having completed whatever work you wished to accomplish, you will now want to shut down the dictionary system.

1. If you have a menu up on the screen, select "0" and exit. If another menu appears, exit in the same manner. You will now have the period prompt on the screen indicating you are out of the dictionary system.
2. Type in the word "QUIT" and hit the RETURN key.
3. If you are using the demonstration system you will get the following prompt:

A>

If you are using the installed system you will get the following prompt:

C>

4. If you were using the installed system, turn off the machine and you are done. If you were using the demonstration system, remove the floppy disks and store them in their paper sleeves.

APPENDIX B  
DATA DICTIONARY SYSTEM'S DATA ELEMENTS

The following is an alphabetic listing of the fields used in the dictionary system. An asterisk following a file name denotes the field being discussed as being key to that file.

DBASE II FIELD NAME / ACRONYM:  aname

FULLNAME: alias name

FORMAT: mixed

LENGTH: 15

FILES WHERE USED: Alias

DESCRIPTION: Name by which any entity is known other than its official name.

DBASE II FIELD NAME / ACRONYM:  bldg

FULLNAME: building

FORMAT: mixed

LENGTH: 15

FILES WHERE USED: Office

DESCRIPTION: The building in which an office is located.

DBASE II FIELD NAME / ACRONYM:  cat

FULLNAME: category

FORMAT: alphabetic

LENGTH: 16

FILES WHERE USED: Category

DESCRIPTION: The category in which an entity would be placed such as a field being a continuation rate.

DBASE II FIELD NAME / ACRONYM: comments

FULLNAME: comments

FORMAT: mixed

LENGTH: 45

FILES WHERE USED: office

DESCRIPTION: Any narrative information that illuminates on the office occurring in the record.

DBASE II FIELD NAME / ACRONYM: datatype

FULLNAME: datatype

FORMAT: alphabetic

LENGTH: 3

FILES WHERE USED: Field

DESCRIPTION: The type of data that a field could hold. The entry can be either A for alphabetic, N for numeric, AN for both or S for special. For example, for a field that could hold all three this entry would be ANS.

DBASE II FIELD NAME / ACRONYM: date

FULLNAME: date

FORMAT: numeric

LENGTH: 6

FILES WHERE USED: Element

DESCRIPTION: This is date an element was created.

dBASE II FIELD NAME / ACRONYM: description

FULLNAME: description

FORMAT: alphabetic

LENGTH: 45

FILES WHERE USED: Report, Model, Program,  
File, Element, Field

DESCRIPTION: Verbal narrative illuminating on the  
entity it is associated with.

dBASE II FIELD NAME / ACRONYM: domain

FULLNAME: domain

FORMAT: mixed

LENGTH: 45

FILES WHERE USED: Field

DESCRIPTION: The range of acceptable values  
a field can take on.

dBASE II FIELD NAME / ACRONYM: editcri

FULLNAME: edit criteria

FORMAT: mixed

LENGTH: 45



FILES WHERE USED: Field

DESCRIPTION: Rules used to constrain and edit what values can occur in a field.

DBASE II FIELD NAME / ACRONYM: elname

FULLNAME: element name

FORMAT: mixed

LENGTH: 12

FILES WHERE USED: Element\*

DESCRIPTION: Unique name of file element (not a data element or field, but the partition of a file).

DBASE II FIELD NAME / ACRONYM: ename

FULLNAME: entity name

FORMAT: mixed

LENGTH: 15

FILES WHERE USED: Alias\*, Category\*, Contains, Calls, Uses, Outputs, Inputs, Alias

DESCRIPTION: The unique name associated with an entity such as what could legally occur in mdl for a Model. This value would be the object of one of the verb relations.

DBASE II FIELD NAME / ACRONYM: ename1

FULLNAME: entity name #1

FORMAT: mixed

LENGTH: 15

FILES WHERE USED: contains\*, Outputs\*, Inputs\*,  
Calls\*

DESCRIPTION: The unique name associated with an  
entity and the subject of a verb relation.

dBASE II FIELD NAME / ACRONYM: etype

FULLNAME: entity type

FORMAT: alphabetic

LENGTH: 10

FILES WHERE USED: Contains, Outputs, Inputs,  
Calls

DESCRIPTION: Type of entity that ename  
specifies such as Model, Report, etc.

dBASE II FIELD NAME / ACRONYM: etype1

FULLNAME: entity type #1

FORMAT: alphabetic

LENGTH: 10

FILES WHERE USED: Contains\*, Outputs\*,Inputs\*,  
Calls\*

DESCRIPTION: Type of entity that ename1 is  
specifying such as Model, Report, etc.

dBASE II FIELD NAME / ACRONYM: fld

FULLNAME: field

FORMAT: mixed

LENGTH: 15

FILES WHERE USED: Field\*

DESCRIPTION: The unique name of a field.

dBASE II FIELD NAME / ACRONYM: fname

FULLNAME: file name

FORMAT: mixed

LENGTH: 16

FILES WHERE USED: File\*

DESCRIPTION: The unique name of a file.

dBASE II FIELD NAME / ACRONYM: fullname

FULLNAME: full name

FORMAT: mixed

LENGTH: 25

FILES WHERE USED: Report, File

DESCRIPTION: The non-unique and fully expanded  
name of either a report or file.

dBASE II FIELD NAME / ACRONYM: language

FULLNAME: language

FORMAT: mixed

LENGTH: 10

FILES WHERE USED: Program

DESCRIPTION: The computer language  
a program is written in, such as  
FORTRAN 77.

dBASE II FIELD NAME / ACRONYM: lstupdate

FULLNAME: last update

FORMAT: numeric, YYMMDD

LENGTH: 6

FILES WHERE USED: File

DESCRIPTION: The most recent date of  
a file having been updated.

dBASE II FIELD NAME / ACRONYM: length

FULLNAME: length

FORMAT: numeric

LENGTH: 3

FILES WHERE USED: Field

DESCRIPTION: The size of a field in bytes.

dBASE II FIELD NAME / ACRONYM: location

FULLNAME: location

FORMAT: mixed

LENGTH: 16

FILES WHERE USED: Model, Program

DESCRIPTION: The file where a program or model

is stored on the computer.

dBASE II FIELD NAME / ACRONYM: mdl

FULLNAME: model

FORMAT: mixed

LENGTH: 8

FILES WHERE USED: Model\*, Maintains\*

DESCRIPTION: The unique abbreviation or acronym associated with a Model.

dBASE II FIELD NAME / ACRONYM: media

FULLNAME: storage media

FORMAT: alphabetic

LENGTH: 10

FILES WHERE USED: File

DESCRIPTION: The type medium used to store a file, i.e. tape or disk.

dBASE II FIELD NAME / ACRONYM: method

FULLNAME: methodology

FORMAT: alphabetic

LENGTH: 45

FILES WHERE USED: Model

DESCRIPTION: The Operations Research solution techniques used by a model such as Linear Programming.



dBASE II FIELD NAME / ACRONYM: modelpoc

FULLNAME: model point of contact

FORMAT: alphabetic

LENGTH: 15

FILES WHERE USED: Maintains

DESCRIPTION: The individual to be contacted  
regarding maintenance of a model.

dBASE II FIELD NAME / ACRONYM: modname

FULLNAME: model name

FORMAT: mixed

LENGTH: 30

FILES WHERE USED: Model

DESCRIPTION: The fully expanded, non-unique  
name of a Model.

dBASE II FIELD NAME / ACRONYM: nullstat

FULLNAME: nullstatus

FORMAT: alphabetic

LENGTH: 1

FILES WHERE USED: Field

DESCRIPTION: Indicates whether a field can have  
the null status or some value must always be  
present.

dBASE II FIELD NAME / ACRONYM: number

FULLNAME: number

FORMAT: mixed

LENGTH: 9

FILES WHERE USED: Office

DESCRIPTION: The number assigned to a report by the computer operations organization of MILPERCEN.

DBASE II FIELD NAME / ACRONYM: cfc

FULLNAME: office symbol

FORMAT: alphabetic

LENGTH: 9

FILES WHERE USED: Office\*, Uses\*

DESCRIPTION: The unique symbol which identifies an office of either DCSPLANS or some other organization, for example PERSINSD.

DBASE II FIELD NAME / ACRONYM: cname

FULLNAME: office name

FORMAT: mixed

LENGTH: 35

FILES WHERE USED: File

DESCRIPTION: The expanded, non-unique name of an office, such Deputy Chief of Staff for Plans for DCSPLANS.

DBASE II FIELD NAME / ACRONYM: outformat

FULLNAME: output format

FORMAT: alphabetic

LENGTH: 15

FILES WHERE USED: Report

DESCRIPTION: The form a report takes such as hard copy, tape, or microfiche.

dBASE II FIELD NAME / ACRONYM: phone

FULLNAME: telephone number

FORMAT: mixed

LENGTH: 21

FILES WHERE USED: Office

DESCRIPTION: The telephone number of an office.

dBASE II FIELD NAME / ACRONYM: pname

FULLNAME: program name

FORMAT: alphabetic

LENGTH: 15

FILES WHERE USED: Program\*

DESCRIPTION: The unique identifier of a program.

dBASE II FIELD NAME / ACRONYM: periodicity

FULLNAME: periodicity of entity

FORMAT: mixed

LENGTH: 4

FILES WHERE USED: Report, Model, Program

DESCRIPTION: How frequently a report, program, or model is executed or produced. Coded by

a number and then the time period. For example,  
2W means twice monthly, 3Q means thrice quarterly.

DBASE II FIELD NAME / ACRONYM: poc

FULLNAME: point of contact

FORMAT: alphabetic

LENGTH: 15

FILES WHERE USED: Office, File, Element, and  
Field

DESCRIPTION: The individual to be contacted if  
having problems or need information about  
the associated entity.

DBASE II FIELD NAME / ACRONYM: purpose

FULLNAME: purpose

FORMAT: alphabetic

LENGTH: 45

FILES WHERE USED: Report, Model, Program,  
File, Field, and Element

DESCRIPTION: The function and reason for  
existence of an entity.

DBASE II FIELD NAME / ACRONYM: rname

FULLNAME: report name

FORMAT: mixed

LENGTH: 10

FILES WHERE USED: Report\*

DESCRIPTION: The unique identifier of a Report.

dBASE II FIELD NAME / ACRONYM: seqnum

FULLNAME: sequence number

FORMAT: numeric

LENGTH: 5

FILES WHERE USED: Element

DESCRIPTION: Location of an element within the file that holds it. Elements are both named and marked by their sequential position in the file.

dBASE II FIELD NAME / ACRONYM: room

FULLNAME: room number

FORMAT: mixed

LENGTH: 6

FILES WHERE USED: Office

DESCRIPTION: Room where an office is located.

dBASE II FIELD NAME / ACRONYM: runame

FULLNAME: run stream name

FORMAT: mixed

LENGTH: 15

FILES WHERE USED: Model

DESCRIPTION: The unique name of a run stream (Job Control Language procedure) that results in Model being executed.



DBASE II FIELD NAME / ACRONYM: size

FULLNAME: size

FORMAT: numeric

LENGTH: 3

FILES WHERE USED: Element

DESCRIPTION: Length in bytes of an occurrence in an element. All occurrences in an element are the same size and the format of the element is likened to a record.

DBASE II FIELD NAME / ACRONYM: seclass

FULLNAME: security classification

FORMAT: alphabetic

LENGTH: 2

FILES WHERE USED: Report, Field

DESCRIPTION: Highest level of classified information the entity contains. Coded as U, C, S or TS.

DBASE II FIELD NAME / ACRONYM: source

FULLNAME: source

FORMAT: mixed

LENGTH: 15

FILES WHERE USED: Field

DESCRIPTION: The document or source which explains or specifies what values can occur in a field. For example, AR 611-201 specifies what would occur in a

field for enlisted military occupational specialty.

DBASE II FIELD NAME / ACRONYM: status

FULLNAME: development status

FORMAT: alphabetic

LENGTH: 15

FILES WHERE USED: Report, Model, Program,  
File, Field

DESCRIPTION: The development status of an entity such as  
test, operational, retired.

DBASE II FIELD NAME / ACRONYM: version

FULLNAME: version

FORMAT: numeric

LENGTH: 3

FILES WHERE USED: Program\*

DESCRIPTION: The edition number of a program and  
therefore an indicator of uniqueness. For example,  
ADSPEC3.

## APPENDIX C

### DATA DICTIONARY SOURCE LISTING

```
* main.prg
* programmer: al noel
* purpose: present main menu for dictionary system
*
*
* SET COLOR TO 30,14
* set talk off
* ERASE
* set up a loop for the user to select which major function to be done
DO WHILE T
  2,0
  TEXT
  This is the MAIN menu for the data dictionary system.
  The system allows you to add, change, or delete records and
  to process queries. Please enter the number of the function you wish
  to perform:
  0 to quit

  1 add a new record
  2 change a record
  3 delete a record
  4 run a query
  5 print a report

  {ADD-MENU}
  {CHG-MENU}
  {DEL-MENU}
  {QUERY-MENU}
  {REPT-MENU}

  Enter the selection here
ENDTEXT
WAIT TO Mselect
* erase error message, if there was any
20,0
21,0
* case statement to select calling of major function menu per user choice
DO CASE
CASE Mselect = 0
  USE
  ERASE
  RELEASE ALL
  RETURN
CASE Mselect = 1
```

```

do add-menu
CASE Mselect = 2
do chg-menu
CASE Mselect = 3
do del-menu
CASE Mselect = 4
do gery-menu
CASE Mselect = 5
do rept-menu
* present error message
OTHERWISE
erase
20,3 SAY Please enter values between 0 and 5 only
21,3 SAY Please try again
ENDCASE
ENDDO
RETURN

```

```

* add-menu.prg
* * programmer: al noel
* * purpose: to present menu for add subsystem

```

```

* * set color to yellow and blue
SET COLOR TO 30,14
SET TALK OFF
* clear screen
ERASE
DO WHILE T
2,0
2, TEXT

```

```

ADD MENU
0 to quit
1 Model (ADD-MODEL)
2 Office (ADD-OFFICE)
3 Report (ADD-REP)
4 Program (ADD-PRO)
5 File (ADD-FI)
6 Element (ADD-EL)
7 Field (ADD-FE)
8 Alias (ADD-AL)
9 Category (ADD-CAT)

```

```

Enter the selection here
ENDTEXT
WAIT TO AMSELECT
20,0 SAY
21,0 SAY

```

```

* case statement to call program based upon users choice
DO CASE
CASE AMSELECT = 0
  CLEAR
  ERASE
  RELEASE ALL
  RETURN
CASE AMSELECT = 1
  DO ADD-MODEL
CASE AMSELECT = 2
  DO ADD-OFFICE
CASE AMSELECT = 3
  DO ADD-REP
CASE AMSELECT = 4
  DO ADD-PRO
CASE AMSELECT = 5
  DO ADD-FI
CASE AMSELECT = 6
  DO ADD-EL
CASE AMSELECT = 7
  DO ADD-FE
CASE AMSELECT = 8
  DO ADD-AL
CASE AMSELECT = 9
  DO ADD-CAT
  * present error message to user
  OTHERWISE
    20,3 SAY PLEASE ENTER VALUES BETWEEN 0 AND 9 ONLY
    21,3 SAY PLEASE TRY AGAIN
  ERASE
ENDCASE
ENDDO
RETURN

```

95

```

* add-model.prg
*
* programmer: al noel
*
* purpose: allow user to add a new model record to dictionary
*
* files used: model, imodel, uses, maintains
*
SET FORMAT TO SCREEN
set exact on
set talk off
ERASE

```



```

REMARK STANDBY PLEASE...
USE MODEL
* index on unique name so can check if user providing new model
INDEX ON MDL TO IMODEL
set color to 30,14
do while t
  erase
  ?
  ?
  ?
  'ADDING NEW MODEL TO DATA DICTIONARY'
  ?
  ?
  Enter the Model's unique name and hit RETURN
  accept To quit hit the space bar once and then RETURN TO AMSHORNAME
  * check to quit
  if amshorname = .OR. AMSHORNAME = .OR. AMSHORNAME =
    erase
    clear
    release all
    return
  endif
  use model index imodel
  * check if model name already exists
  find &amshorname
  if <> 0
    erase
    ?
    ?
    ? A record for the Model named , amshorname, already exists
    store 1 to xx
    do while xx < 45
      store xx+1 to xx
    enddo
  ELSE
    erase
    * initialize variables for use in screen
    store to ammdl
    store to ammodname
    store to ammeth
    store to amstatus
    store to amallocation
    store to ampurpose
    store to amdescrip
    store to amruname
    store t to xflag
    * present screen
    do while xflag
      2,3 say Model unique name get ammdl
      4,3 say Model full name get ammodname
    enddo
  enddo
enddo

```

```

6,3 say Methodology get ammeth
8,3 say Status get amstatus
10,3 say Location get amlocation
12,3 say Purpose get ampurpose
14,3 say Description get amdescrip
16,3 say Runstream get amruname
18,3 say Periodicity get amperiod
read store f to errflag
if .not. errflag
  store f to xflag
endif
enddo while xflag
* create new blank record to load
* append blank
* load new record from whats on screen
replace mdl with !(ammdl), mcdname with !(ammodname)
replace location with !(amlocation), purpose with !(ampurpose)
replace descrip with !(amdescrip), runame with !(amruname)
replace period with !(amperiod), status with !(amstatus)
replace method with !(ammeth)
erase t to offlag
store t to offlag
do while offlag
  ? Enter the symbol for the office using this model and RETURN to amoff
  accept To quit entering offices hit the space bar and RETURN to amoff

  *check to quit
  if amoff=.or. amoff = .or. amoff =
    store f to offlag
    erase
  else
    use uses
    append blank
    replace ofc with !(amoff), ename with !(ammdl), etype with MODEL
    erase
  endif
enddo while offlag
* store maintainers
store t to flag
do while flag
  ? Enter office symbol for office maintaining this model and RETURN
  accept To quit entering maintainers hit space bar and RETURN to amoff
  if amoff=.or. amoff = .or. amoff =
    store f to flag
    erase

```

```

else
accept Who is the point of contact ? to ampoc
use maintains
append blank
replace ofc with !(amcff), mdl with !(ammdl)
replace modelpoc with !(ampoc)
erase
endif
enddo while flag
ENDIF
erase
enddo

```

```

* add-office.prg
**
** programmer: al noel
**
** purpose: to permit user to add a new office record to dictionary
**
** files used: office, ioffice
**
SET FORMAT TO SCREEN
set exact on
set talk off
set color to 30,14
ERASE
REMARK STANDBY PLEASE ...
*index file so can check if record with unique office symbol already exists
USE OFFICE
INDEX ON OFC TO IOFFICE
USE OFFICE INDEX IOFFICE
do while t
erase
??
??
* ? ? Enter the Office's unique symbol and hit RETURN
accept To quit hit the space bar once and then RETURN TO AMSHORNAME
* check to quit, if blank quit
if amshorname = .OR. AMSHORNAME = .OR. AMSHORNAME =
erase
clear

```

```

release all
return
endif
* check if office symbol user gives is truly unique & therefore addable
Find &amshorname
if
<> 0
erase
?
?
? ? A record for an Office with symbol , amshorname, already exists
store 1 to xx
do while xx < 45
store xx+1 to xx
enddo
ELSE
erase
* initialized screen variables
store to amofc
store to amroom
store to ambldg
store to amphone
store to ampoc
store t to xflag
do while xflag
* put up screen
2,3 say Office's unique symbol get amofc
4,3 say Office's full name get amoname
6,3 say Building get ambldg
8,3 say Room get amroom
10,3 say Phone number get amphone
12,3 say Comments get amccmnts
14,3 say Point of Contact get ampoc
read
store f to errflag
if .not. errflag
store f to xflag
endif
enddo while xflag
* create new blank record and load with data from screen
append blank
replace ofc with !(amofc), oname with !(amname)
replace bldg with !(ambldg), room with !(amroom)
replace phone with !(amphone), comments with !(amcomments)

```

```

replace poc      with !(ampoc )

erase
ENDIF
erase
enddo

** add-alias.prg
** purpose: to add a new alias record to dictionary
** programmer : al noel
** files used: alias, ialias
SET FORMAT TO SCREEN
set exact on
set talk off
set color to 30,14
ERASE
REMARK STANDBY PLEASE...
* index so can check for uniqueness of user's entry for name
USE ALIAS
INDEX ON ENAME - ANAME TO IALIAS
USE ALIAS INDEX IALIAS
do while t
erase
? ? ?
? ? ?
? ? ?
? ? ?
accept Enter the Entity's unique name and hit RETURN to part1
accept Enter the Entity's alias and hit RETURN to part2
* check to quit
if part2 = .OR. part2 =
erase
clear
release all
return
endif
store part1 - part2 to key
* check for uniqueness of entry given by user
find &key
if

```



```

<> 0  erase
    ?
    ?
? A record for an Entity named ,part1, with alias of      ,part2
  already exists
  store 1 to xx
  do while xx < 45
    store xx+1 to xx
  enddo
ELSE
  erase
  * init screen variables  to amename
  store part1
  store part2          to amename
  store                to ametype
  store t to xflag
  do while xflag
  * put up screen
    2,3 say Entity's unique name get amename
    4,3 say Alias get amaname
    6,3 say Type of entity (i.e. MODEL, or REPORT) get ametype
  read
    store f to errflag
    if .not. errflag
      store f to xflag
    endif
  enddo while xflag
  * create new blank record and load with data from screen
  append blank
  replace ename with !(amename), aname  with !(amename)
  replace etype with !(ametype)
  erase
ENDIF
erase
enddo

```

101

```

* add-report.prg
* *
* programmer: al noel
* *
* purpose: to permit user to add a new report record to dictionary

```



```

store
store      to amnumber
store      to amstatus
store      to amperiod
store      to amseclss   to amoutfrmat
store      to amoutfrmat

store t to xflag

do while xflag
# present screen
2,3 say Report's full name get amrname amfullname
4,3 say Report's purpose get ampurpose
6,3 say Description get amdescrip
8,3 say Number get amnumber
10,3 say Status get amstatus
12,3 say Periodicity get amperiod
14,3 say Security classification get
16,3 say Output format get amoutfrmat
18,3 say amseclss
read
store f to errflag
if .not. errflag
    store f to xflag
endif
enddo while xflag
* create new blank record and load with data from screen
append blank
replace rname with !(amrname), fullname with !(amfullname)
replace purpose with !(ampurpose), descrip with !(amdescrip)
replace number with !(amnumber), status with !(amstatus)
replace period with !(amperiod), seclss with !(amseclss)
replace outfrmat with !(amoutfrmat)
erase
store t to offlag
do while offlag
    Enter the symbol for the Office which uses this Report
    and then hit RETURN
? ?
? ?
* put intersection record in for office using report just entered
* if blank then quit this phase of program
accept To quit entering Offices hit the space bar and RETURN to amoff
if amoff = .or. amoff =
    store f to offlag
    erase
else
    use uses

```

```

append blank      with !(amof),  ename  with !(amrname)
replace ofc       with REPORT
replace etype     with REPORT
erase
endif
enddo while offlag
erase
store t to flag
* put in intersection record for models outputting report
do while flag
? ? Enter the unique name for the Model that outputs this Report
? ? and then hit RETURN
* check to quit entering models
accept if amof = .or. amof = .or. amof =
clear
store f to flag
erase
else
use outputs
append blank
replace ename1 with !(amof), etype1 with MODEL
replace ename with !(amrname), etype with REPORT
erase
endif
enddo while flag

ENDIF
erase
enddo

```

104

```

* add-field.prg
* * purpose: to add a new field to the dictionary
* * programmer: al noel
* * files used: field, ifield, contains
* *
set exact on
SET FORMAT TO SCREEN
set talk off
set color to 30,14

```

```

ERASE
REMARK STANDBY PLEASE...
* index on unique name so can check for unique new name coming from user
USE FIELD
INDEX ON FLD TO IFIELD
do while t
USE FIELD INDEX IFIELD
erase
? ? 'ADDING NEW FIELD TO THE DATA DICTIONARY'
? ?
? ? Enter the Field's unique name and hit RETURN
* check to quit
accept To quit hit the space bar once and then RETURN TO AMSHORNAME
if amshorname = .OR. AMSHORNAME = .OR. AMSHORNAME =
erase
clear
release all
return
endif
* check for unique name already existing, if so, tell user
Find &amshorname
if
<> 0
erase
? ?
? ? ? A record for a Field with the name , amshorname, already exists
store 1 to xx
do while xx < 45
store xx+1 to xx
enddo
ELSE
erase
* init screen variables
store to amfld
store to amlength
store to amdatatype
store to amnullstat
store to amsource
store to amseclass
store to amstatus
store to ampoc
store t to xflag

```

to amdomain  
to ampurpose  
to amdescrip



```

do while xflag
  set format to fedat.fmt
  read
  store f to errflag
  if .not. errflag
    store f to xflag
  endif
enddo while xflag
* create new blank record and load with data from screen
append fld with !(amfld), length with !(amlength)
replace datatype with !(amdatatype), nullstat with !(amnullstat)
replace source with !(amsource), domain with !(amdomain)
replace purpose with !(ampurpose), descrip with !(amdescrip)
replace status with !(amstatus), seclass with !(amseclass)
replace poc with !(ampoc)
erase
store T to offlag
do while offlag
  erase
  ?? 'Enter the unique name for the Element which contains this Field'
  ??
  ??
  accept 'To quit entering Elements hit the space bar and RETURN' to amoff
  * check to quit entering elements that contain field
  if amoff = ' '.or. amoff = ' '.or. amoff = ' '
    clear
    store f to offlag
    erase
  else
    * put in intersection record in contains file for new field
    use contains
    append blank
    replace ename1 with !(amoff), etype1 with 'ELEMENT'
    replace ename with !(amfld), etype with 'FIELD'
  endif
enddo while offlag
erase
ENDIF
erase
enddo

* add-program. prg

```

```

**** purpose: to add a new program record to dictionary
**** programmer: al noel
**** files used: program, iprogram
set exact on
SET FORMAT TO SCREEN
set talk off
set color to 30,14
ERASE
**** index file so can check for uniqueness of entry given by user
remark STANDBY WHILE I INDEX FILE PLEASE
use program
index on pname - version to iprogram
**** set up loop to do work
do while t
erase
? ? ? ? ?
? ? ? ? ? To quit hit space and RETURN for the following two prompts
? ? ? ? ?
accept Enter the Program's unique name and hit RETURN to part1
accept Enter the Program's version number and hit RETURN to part2
check to quit or do work
if part2 = .OR. part2 = .OR. part2 =
erase
clear
release all
return
endif
use program index iprogram
store part1 - part2 to key
look for record with unique name and version as given by user
****
Find & key
if no find, tell user and ask if he wishes to continue
if 0
<> erase
? ?
? ?
? ? A record for a Program named ,part1, with version number, part2
already exists
store 1 to xx

```

```

do while xx < 45
  store xx+1 to xx
enddo
ELSE
  erase
  * init screen variables
  store to ampname
  store to amlanguage
  store to amversion
  store to amstatus
  store t to xflag
  store t to xflag
do while xflag
  * put up screen for user to load with data
  2,3 say Program's unique name get ampname
  4,3 say Location get amlocation
  6,3 say Version get amversion
  8,3 say Status get amstatus
  10,3 say Purpose get ampurpose
  12,3 say Description get amdescrip
  14,3 say Periodicity get amperiod
  16,3 say Language get amlanguage
  read
  store f to errflag
  if .not. errflag
    store f to xflag
  endif
enddo while xflag
  use program index iprogram
  * create new blank record and load with data from screen user gave u
  append blank
  replace pname with !(ampname ), location with !(amlocation)
  replace version with !(amversion ), purpose with !(ampurpose)
  replace descrip with !(amdescrip), language with !(amlanguage)
  replace period with !(amperiod), status with !(amstatus)
erase
store t to offlag
do while offlag
  * prompt to build calls record of model that calls program just entered
  ?? Enter the unique name for the Model which calls this program
  ?? and then hit RETURN
  ??

```

```

accept To quit entering Models hit the space bar and RETURN to amoff

if amoff = .or. amoff = .or. amoff =
clear
store f to offlag
erase
else
use calls
store ampname - amversion to holder
append blank
replace ename1 with !(amoff), etype1 with MODEL
replace ename with !(holder }, etype with PROGRAM
erase

endif
enddo while offlag
ENDIF
erase
enddo

* add-file.prg
* purpose: to add a new file to the data dictionary
* programmer: al noel
* file used: file, ifile, contains, inputs, outputs
SET FORMAT TO SCREEN
set exact on
set talk off
set color to 30,14
ERASE
REMARK STANDBY PLEASE...
* index so can check for unique file name by user
USE FILE
INDEX ON FNAME TO IFILE
do while t
USE FILE INDEX IFILE
erase
??
? ?
? ? To quit hit space bar and then RETURN for the following prompt

```

```

? ?
accept Enter the File's unique name and then hit RETURN to part1
* check to quit
if part1 = .OR. part1 = .OR. part1 =
erase
clear
release all
return
endif
* check if name given by user is unique , if not tell him so
Find %part1
if
<> 0 erase
? ?
? A record for a File named ,part1, already exists
store 1 to xx
do while xx < 55
store xx+1 to xx
enddo
ELSE
erase
* init screen variables to amfname
store PART1 to amfname
store to ammedia
store to amseclass
store to amstatus
store ' ' to amlstupdat
store t to xflag
do while xflag
* est. screen
set format to fidat.fmt
read
store f to errflag
if .not. errflag
store f to xflag
endif
enddo while xflag
* create new blank record and load with data from screen
append blank
replace fname with !(amfname ), fullname with !(amfullname)

```

```

replace media with !(ammedia ), status with !(amstatus )
replace descrip with !(amdescrip), purpose with !(ampurpose )
replace poc with !(ampoc ) { seclass with !(amseclass)
erase lstupdat with amlstupdat
store t to offlag
do while offlag
? Enter the unique name for a Model that receives input from this File
? and then hit RETURN
??
* create intersection records for inputs file
accept To quit entering Models hit the space bar and RETURN to amof

if amof = .or. amof = .or. amof =
store f to offlag
erase
else
use inputs
append blank
replace ename1 with !(amof), etype1 with MODEL
replace ename with !(amfname), etype with FILE
erase
endif
enddo while offlag

erase
store t to flag
do while flag
? Enter the unique name for a Model that outputs to this File
? and then hit RETURN
??
* create intersection record for outputs file
accept To quit entering Models hit the space bar and RETURN to amof
if amof = .or. amof = .or. amof =
clear
store f to flag
erase
else
use outputs
append blank
replace ename1 with !(amof), etype1 with MODEL
replace ename with !(amfname), etype with FILE
erase
ENDIF
enddo while flag

```



```
endif
erase
enddo
```

```
* add-element.prg
** purpose: to add a new element to the dictionary
** programmer: al noe
** files used: element, ielement, contains
set exact on
SET FORMAT TO SCREEN
set talk off
set color to 30,14
ERASE
REMARK STANDBY PLEASE...
** index so can check for uniqueness
USE ELEMENT
INDEX ON ELNAME TO IELEMENT
do while t
  USE ELEMENT INDEX IELEMENT
  erase
  ??
  ??
  ??
  ??
  ??
  To quit hit the space bar and RETURN
accept Enter the Element's unique name and hit RETURN to part1
  * check to quit using program = .OR. part1 =
  if part1 = .OR. part1
    erase
    clear
    release all
    return
  endif
  store part1 to key
  * check for uniqueness of entry given by user
  find &key
  if
    <> 0
  erase
```

```

??
? A record for a Element named ,part1, already exists
  store 1 to xx
  do while xx < 55
    store xx+1 to xx
  enddo
ELSE
  erase
  * init screen variables
  store to amename
  store to amversion
  store to amdate
  store to amsegnum
  store to amsize
  store to ampoc
  store t to xflag

  do while xflag
    * put up screen so user can load data
    set format to eldat
    read
    store f to errflag
    if .not. errflag
      store f to xflag
    endif
  enddo
  while xflag
    append blank
    * load record with data from screen
    replace elname with !(amename), version with !(amversion)
    replace date with !(amdate), segnum with !(amsegnum)
    replace size with !(amsize), poc with !(ampoc)
    replace purpose with !(ampurpose), descrip with !(amdescrip)
  erase
  store t to offlag
  do while offlag
    * Enter the unique name for the File which contains this element
    and then hit RETURN
  ??
  ??
  * add intersection record for contains file
  accept to quit entering Files hit the space bar and RETURN to amoff
  if amoff = .or. amoff = .or. amoff =
    clear
    store f to offlag

```

to ampurpose  
to amdescrip

```

        erase
    else
        use contains
        append blank
        replace ename1 with !(amoff), etype1 with FILE
        replace ename with !(amelname), etype with ELEMENT
        erase
    endif
enddo while offlag
ENDIF
erase
enddo

```

```

* add-category.prg
* *
* * purpose: to add a new category record to the dictionary
* *
* * programmer: al noel
* *
* * files used: category, icategory
*
set exact on
set FORMAT TO SCREEN
set talk off
set color to 30,14
ERASE
REMARK STANDBY PLEASE
* index so can check for uniqueness of new entry by user
USE CATEGORY
INDEX ON ENAME - ETYPE TO ICATEGORY
USE CATEGORY INDEX ICATEGORY
do while .t
    erase
    ? ? ?
    ? ? ? 'ADDING NEW CATEGORY TO DATA DICTIONARY'
    ? ? ?
    ? ? ? To quit hit space and RETURN for the following two prompts
    ? ? ?
accept Enter the Entity's unique name and hit RETURN to part1
accept Enter the Entity's type (i.e. MODEL) and hit RETURN to part2
* check to quit, if blank quit, else, go on
if part2 = .OR. part2 = .OR. part2 =
    erase

```

```

clear
release all
return
endif
store part1 - part2 to key
* check for uniqueness, if not unique, tell user and don't allow entry
find &key
if
<> 0
erase
?
?
? A record for an Entity named ,part1, with a type of ,part2
already exists
store 1 to xx
do while xx < 45
store xx+1 to xx
enddo
ELSE
erase
* init screen variables to amename to ametype
store part1
store part2
store to amcat
store t to xflag
* put up screen for user to load data into
do while xflag
set format to catdat
read store f to errflag
if .not. errflag
store f to xflag
endif
enddo while xflag
* create new blank record and load with data from screen
append blank
replace ename with !(amename), cat with !(amcat)
replace etype with !(ametype)
erase
ENDIF
erase
enddo

```

```

* PROGRAM CHG-MENU
*
* purpose: permit user to select change program he wishes to execute
*
* programmer: al noel
*
SET COLOR TO 30,14
SET TTY OFF
* CLEAR SCREEN
* SET UP DO LOOP FOR SELECTIONS
DO WHILE T
  2,0
  TEXT
  This is the CHANGE menu. It permits you to change records in
  any file. Please enter the number of the function you wish to
  perform.
  0 to quit
  1 Model {CHG-MODEL}
  2 Office {CHG-OFFICE}
  3 Report {CHG-REP}
  4 Program {CHG-PRO}
  5 File {CHG-FI}
  6 Element {CHG-EL}
  7 Field {CHG-FE}
  8 Alias {CHG-AL}
  9 Category {CHG-CAT}

```

```

Enter the selection here
ENDTEXT
WAIT TO CMSELECT
20,0 SAY
21,0 SAY
* CASE STATEMENT TO CALL PROGRAM BASED UPON USERS CHOICE
DO CASE CMSELECT = 0
  USE
  ERASE
  RELEASE ALL
  RETURN
  CASE CMSELECT = 1
    DO CHG-MODEL
  CASE CMSELECT = 2
    DO CHG-OFFICE
  CASE CMSELECT = 3
    DO CHG-REP
  CASE CMSELECT = 4
    DO CHG-PRO

```

```

CASE CMSELECT = 5
  DO CHG-FI
CASE CMSELECT = 6
  DO CHG-EL
CASE CMSELECT = 7
  DO CHG-FE
CASE CMSELECT = 8
  DO CHG-AL
CASE CMSELECT = 9
  DO CHG-CAT
* error message if wrong number pushed by user
  OTHERWISE
    ERASE
      20,3 SAY Please enter values between 0 and 9 only
      21,3 SAY Please try again
ENDCASE
ENDDO
RETURN

```

```

* Program   CHG-FILE
** purpose: to permit user to change a file type record
**
** programmer: al noel
**
** files used: file, ifile, inputs, outputs
**
** programs called: afiimod, dfiimod, afiomod, dfiomod
set exact on
SET FORMAT TO SCREEN
SET COLOR TO 30,14
SET TALK OFF
ERASE
REMARK STANDBY PLEASE...
USE FILE INDEX IF FILE
* set up loop to do work
STORE t TO cmmore
DO WHILE CMMORE
  ERASE
    ?
  accept Enter the unique name of the File  you wish to change  TO CMNUM
  ERASE
  * look for record with unique name given by user
  FIND &CMNUM
  * if cant find record with unique name, tell user

```



```

IF = 0 ERASE
?
?
?
  A Record with File named CMNUM, is not in the file
ACCEPT Enter a Y to try again or N if not and then hit RETURN to cmansw
* check if user wants to try again or quit program
IF CMANSW <> Y .AND. CMANSW <> Y
  STORE F TO CMCORE
ENDIF
ELSE
  * store file data into memory variables
  store fname to oldfname
  store fname to amfname
  store fullname to amfullname
  store purpose to ampurpose
  store descrip to amdescrip
  store status to amstatus
  store seclass to amseclass
  store poc to ampoc
  store media to ammedia
  store lstupdat to amlstupdat
  * set up screen for user to edit data
  set format to fidat
READ
* replace data in file with edited data from screen
  REPLACE fname WITH !(amfname), fullname WITH !(amfullname)
  REPLACE purpose WITH !(ampurpose), descrip WITH !(amdescrip)
  REPLACE poc WITH !(ampoc), status WITH !(ampoc)
  replace seclass with !(amseclass), media with !(ammedia)
  replace lstupdat with amlstupdat
  * if unique name of file changed, change associate files of inputs& outputs
  if
    y(amfname) <> oldfname
    ERASE
    14,3 SAY 'STANDBY PLEASE...'
    use inputs
    do while .not. eof
      if TRIM(ename) = TRIM( oldfname) .and. ETYPE = 'FILE'
        replace ename with !(amfname)
      endif
    skip
    enddo
    use outputs
  do while .not. eof
    if TRIM(ENAME) = TRIM(oldfname) .and. etype = 'FILE'

```

```

        replace ename with !(amfname)
    endif
    skip
enddo
* if new unique name of file, then change in contains file
use contains
do while .not. eof
    if trim(ename1) = trim(oldfname) .and. trim(etype1) = 'FILE'
        replace ename1 with !(amfname)
    endif
    skip
enddo
endif
* check if user needs to add models that input from this file
? Do you need to add new models that inputs this file?
accept 'If so, hit Y and RETURN, else hit space bar and RETURN to answer
if 'Y' = !(answer)
    ERASE
    do afiimod
    endif
    erase
? ? ?
? 'Do you wish to delete a Model that inputs this file ?'
accept 'If so enter a Y and RETURN, else hit the space bar and RETURN' to answer
* check of see if user needs to delete from inputs file models that input file
if 'Y' = !(answr)
    erase
    do dfiimod
    endif
    erase
? ? ?
? 'Do you wish to add a Model that outputs this file ?'
accept 'If so enter an Y and RETURN, else hit space bar and RETURN' TO ANSWR
* check if user wants to add outputs record that model outputs this file
IF 'Y' = !(ANSWR)
    ERASE
    DO AFiOMOD
    ENDIF
    erase
? ? ?
? 'Do you wish to delete a Model that outputs this file ?'
accept 'If user wants to delete a model from outputs file for edited file
if 'Y' = !(answr)

```

```

erase
do dfiomod
endif
  ERASE
  ?
  ?
  ?
  * prompt user for choice to change another record or to leave pgm
  ACCEPT ENTER A Y IF YOU WANT TO CHANGE ANOTHER FILE TO CMCONT
  ERASE
  IF CMCONT <> Y .AND. CMCONT <> Y
    ERASE
    STORE F TO CMMORE
  ELSE
    USE FILE INDEX IF FILE
  ENDIF
ENDIF

*END OF DO WHILE
ENDDO
ERASE
RELEASE ALL
RETURN

```

```

* afimod.prg
*
* purpose: to add model to inputs file for a preexisting file
*
* programmer: al noe
*
* files used: inputs
  ERASE
  ?
  ?
  accept 'Enter unique name of Model and hit RETURN' to mod
  use inputs
  append blank
  replace ename1 with !(mod), etype1 with 'MODEL'
  replace ename with !(amfname), etype with 'FILE'
  erase
  return
* dfimod.prg delete models in inputs file that use a preexisting file

```

```

** purpose: delete models inputs file that no longer input from preexisting file
** programmer: al noel
** files used: inputs
erase
?
?
accept 'Enter the unique name of the Model and hit RETURN' to mod
erase
14,3 say 'Standby ...'
use inputs
store !(amfname) to amfname
store !(mod) to mod
do while .not. eof
if TRIM(ename1) = TRIM(mod) .and. TRIM(etype1) = 'MODEL'
.and. trim(amfname) = trim(amfname) .and. etype = 'FILE'
delete
endif
skip
enddo
pack
return
** afiomod.prg to add model that outputs a preexisting file
** purpose: to add model that outputs a preexisting file
** programmer: al noel
** files used: outputs
?
?
accept 'Enter unique name of Model and hit RETURN' to mod
use outputs
append blank
replace ename1 with !(mod) , etype1 with 'MODEL'
replace ename with !(amfname), etype with 'FILE'
erase
return
** dfiomod.prg delete models in output file that produce a preexisting file
** purpose: delete models in output file that produce a preexisting file
** programmer: al noel

```

```

** files used: outputs
** erase
** ?
accept 'Enter the unique name of the Model and hit RETURN' to mod
erase
14,3 say 'Standby ...'
use outputs
store !(amfname) to amfname
store !(mod) to mod
do while .not. eof
if trim(enamei) = trim(mod) .and. trim(etype1) = 'MODEL'.and.
trim(ename) = trim(amfname) .and. trim(etype) = 'FILE' ;
delete
endif
skip
enddo
pack
return
** Program CHG-FIELD
** purpose: to change field record
** programmer: al noel
** { used: field, ifield
** programs called: afeel, dfeel
**
set exact on
SET FORMAT TO SCREEN
SET COLOR TO 30,14
SET TALK OFF
ERASE
REMARK STANDBY PLEASE...
USE FIELD INDEX IFIELD
* set up loop to do work
STORE t to cmmore
DO WHILE CMMORE
ERASE
? ?
accept 'Enter the unique name of the Field you wish to change to CMNUM
ERASE
* locate record with unique name given by user
FIND &CMNUM
* if no find, tell user and allow to quit or not

```

```

IF = 0 ERASE
    ?
    ?
    ? A Record with Field named CMNUM, is not in the file
    ACCEPT Enter a Y to try again or N if not and then hit RETURN to cmansw
    IF CMANSW <> Y -AND. CMANSW <> Y
        STORE F TO CMMORE
    ENDIF
ELSE
    * store file data into memory variables
    store fld to oldfld
    store fld to amfld
    store length to amlength
    store datatype to amdatatype
    store nullstat to amnullstat
    store source to amsource
    store domain to amdomain
    store purpose to ampurpose
    store descrip to amdescrip
    store status to amstatus
    store seclass to amseclass
    store poc to ampoc
    * set up screen for user to edit data
    set format to fedat

READ
* REPLACE data in file with data from screen with ! (amlength)
REPLACE fld with ! (amfld), length with ! (amnullstat)
REPLACE datatype WITH ! (amdatatype), nullstat WITH ! (amdomain)
REPLACE source WITH ! (amsource), descrip with ! (amdescrip)
REPLACE purpose with ! (ampurpose), poc with ! (ampoc), seclass with ! (amseclass)
REPLACE status with ! (amstatus), poc with ! (ampoc), then change that in contains file
* if new unique name for field, then change that in contains file
if amfld <> oldfld
    ERASE
    14,3 SAY 'STANBY PLEASE...'
    use contains
    do while .not. eof
        if TRIM(ename) = TRIM(oldfld) -and. trim(etype) = 'FIELD'
            replace ename with ! (amfld)
        endif
        skip
    enddo
endif
ERASE
? Do you need to add new elements that contain this field ?

```



```

* check if user needs to add new elements to contains file for this field
accept If so, hit Y and RETURN, else hit space bar and RETURN to answer
if 'Y' = !(answer)
  ERASE
  do afeel
endif
ERASE

? * check if user needs to delete from contains file for this particular field
accept 'Delete elements which no longer contain this field ?,Y or space' to an
if 'Y' = !(an )
  erase
  do dfeel
endif
ERASE
? ? ?

* prompt user for choice to change another record or to leave pgm
ACCEPT ENTER A Y IF YOU WANT TO CHANGE ANOTHER FIELD TO CMCONT
ERASE
IF CMCONT <> Y .AND. CMCCNT <> Y
  STORE F TO CMMORE
ELSE
  USE FIELD INDEX IFIELD
ENDIF
ENDIF

*END OF DO WHILE
ENDDO
ERASE
RELEASE ALL
RETURN

```

124

```

* afeel.prg      to add elements (containers) of fields that preexist
**
** purpose: to add elements(containers) of fields that preexist
**
** programmer: al noel
**
** files used: contains
**

```

```

store t to offlag
do while offlag
  ERASE
  ?
  ?
  ? Enter the name for the element containing this field and RETURN
  accept 'To quit entering elements hit the space bar and RETURN to amoff
  * check to quit or not
  if amoff = '.or. amoff = ' ' .or. amoff = ' '
    store f to offlag
    erase
    return
  else
    * put new record in contains file for field and element given by user
    use contains
    append blank
    replace ename1 with !(amoff), etype1 with 'ELEMENT'
    replace ename with !(amfld), etype with 'FIELD'
  endif
enddo while offlag
* dfeel.yig
* * * * * purpose: to remove records from contains file where element containing
* * * * * a field should no longer bekep as containing the field
* * * * * programmer: al noel
* * * * * files used: contains
store t to flag
do while flag
  erase
  ?
  ? Enter name for element no longer containing this field and RETURN
  accept 'To quit deleting elements hit space bar and RETURN' to amoff
  * check to quit or continue on
  if amoff = ' ' .or. amoff = ' ' .or. amoff = ' '
    store f to flag
    use contains
    pack
    erase
    return
  else
    erase
    14,3 say 'Standby please....'
    * delete all records that meet criteria
    use contains
    do while .not. eof

```

```

if trim(ename) = trim(amfld) .and. trim(etype) = 'FIELD' .and. ;
    delete
endif
skip
enddo
endif
enddo while flag

* Program CHG-MODEL
** purpose: to permit user to change record for a model
** programmer: al noel
** files used: model, imodel,
** programs called: adoff, doff, admain, dmain
set exact on
SET FORMAT TO SCREEN
SET COLOR TO 30,14
SET TALK OFF
ERASE 77 SAY BE PATIENT...
USE MODEL INDEX IMODEL
*set up loop to do work
STORE t TO cmmore
DO WHILE CMMORE
    ERASE
    ?
    ?
accept Enter the unique name of the Model you wish to change TO CMNUM
ERASE
* locate record for unique name given by user
FIND &CMNUM
*if no find, tell user and ask if wasnt to continue or quit program
IF = 0
    ERASE
    ?
    ?
    ? A Record with Model name CMNUM is not in the file
    ACCEPT Enter a Y to try again or N if not and then hit RETURN to cmansw
    IF CMANSW <> Y .AND. CMANSW <> Y
        STORE F TO CMMORE
    ENDIF
ELSE

```

```

* store file data into memory variables for editing in screen
store mdl to oldmdl
store mdl to cmdl
store modname to cmmodname
store method to cmmethod
store status to cmstatus
STORE location TO Cmlocation
STORE purpose TO Cmpurpose
store descrip to cmdescrip
store runame to cmruname
store period to cmperiod
* put up screen for user to edit data
2,1 SAY Unique name of Model get cmdl
4,1 SAY Full name of Model get cmmodname
6,1 SAY Methodology used get cmmethod
8,1 SAY Status get cmstatus
10,1 SAY Location get cmlocation
12,1 SAY Purpose get cmpurpose
14,1 SAY Description get cmdescrip
16,1 SAY Runstream get cmruname
18,1 SAY Periodicity get cmperiod

READ
* replace data in file with edited data from screen
REPLACE method WITH !(Cmethod), modname with !(cmmodname)
REPLACE location WITH !(Cmlocation), status with !(cmstatus)
REPLACE descrip WITH !(cmdescrip), purpose WITH !(cmpurpose)
replace period with !(cmperiod), runame with !(cmruname)
* if unique name was changed, then change uses, maintains, inputs, outputs
* to new unique name
if !(cmdl) <> oldmdl
14,3 SAY 'STANDBY PLEASE...'
use uses
do while .not. eof
if ename = oldmdl
replace ename with !(cmdl)
endif
skip
enddo
use maintains
do while .not. eof
if mdl = oldmdl
replace mdl with !(cmdl)
endif
skip
enddo

```

```

use calls
do while .not. eof
  if trim(ename1) = TRIM(cldmdl) .and. trim(etype1) = 'MODEL'
    replace ename1 with !(cmdl)
  endif
  skip
enddo
use outputs
do while .not. eof
  if trim(ename1) = trim(oldmdl) .and. trim(etype1) = 'MODEL'
    replace ename1 with !(cmdl)
  endif
  skip
enddo
use inputs
do while .not. eof
  if trim(ename1) = trim(oldmdl) .and. trim(etype1) = 'MODEL'
    replace ename1 with !(cmdl)
  endif
  skip
enddo
endif
erase
? Do you need to add new users of this Model ?
accept if so, hit Y and RETURN, else hit space bar and RETURN to answer
*check if user needs to add new users of model to uses file
if 'Y' = !(answer)
  ERASE
  do adoff
  endif
  erase
?
accept 'Do you wish to delete any users Y or N' to answer
*check if user needs to delete users of model
if 'Y' = !(answer)
  erase
  do doff
  endif
  erase
* check if needs to add new maintainers of model
? Do you need to add new maintainers of this Model ?
accept if so, hit Y and RETURN, else hit space bar and RETURN to answer
if 'Y' = !(answer)
  ERASE
  do admain
  endif
  ERASE
?

```

```

? 'Do you need to delete any maintainers of this Model?'
*check if user needs to delete maintainers of model
accept 'If so, hit Y and RETURN, else hit space bar and RETURN' to answer
if 'Y' = ' (answer)
    erase
    do dmain
    endif
    ERASE
    ?
    ?
* prompt user for choice to change another record or to leave pgm
ACCEPT ENTER A Y 'If YOU WANT TO CHANGE ANOTHER MODEL TO CMCONT
ERASE
IF CMCONT <> Y .AND. CMCCNT <> Y
    STORE F TO CMMORE
ELSE
    USE MODEL INDEX IMODEL
ENDIF
ENDIF

*END OF DO WHILE
ENDDO
ERASE
CLEAR
RELEASE ALL
RETURN

** program: adoff.prg
** purpose: to add offices that use an existing model
** programmer: al noel
** files used: uses
store t to offlag
do while offlag
    ? Enter the symbol for the office using this model and RETURN
    accept To quit entering offices hit the space bar and RETURN to amoff
    if amoff = '.or.' .or. amoff = ',.or. amoff = '
        store f to offlag

```



```

erase
return
else
  use uses
  append blank
  replace ofc with !(amoff), ename with !(cmdl), etype with 'MODEL'
  erase
ENDIF
ENDDO WHILE OFFLAG
** program: doff.prg
** purpose: to delete offices from uses file that no longer use existing model
** programmer: al noel
** files used: uses
store t to flag
do while flag
?
? 'Enter symbol for office no longer using this Model and RETURN'
accept 'To quit deleting users hit space bar and RETURN' to amoff
* check if user wishes to quit or continue
if amoff = ' ' .or. amoff = ' ' .or. amoff = ' '
  store f to flag
  use uses
  * actually remove records from uses file
  pack
  erase
  return
else
  erase
  14,3 say 'Standby please....'
  * mark records for deletion if are of model being changed and office given
  * by user to delete
  use uses
  do while .not. eof
    if ename = cmdl .and. ofc = amoff .and. etype = 'MODEL'
      delete
    endif
    skip
  enddo
  endif
enddo while flag
** program ADMMAIN.PRG
** purpose: to add new maintainers (offices) of an existing model
**

```

```

* programmer: al noel
** files used: maintains
*
store t to flag
do while flag
? 'Enter office symbol for office maintaining this model and RETURN'
accept 'To quit entering maintainers hit space bar and RETURN' TO AMOFF
* check if user wants to quit or continue
IF AMOFF = ' ' .OR. AMOFF = ' ' .OR. AMOFF = ' '
STORE F TO FLAG
ERASE
RETURN
ELSE ACCEPT 'Who is the point of contact ?' to ampoc
* create new record and put in name given by user and model that was edited
use maintains
append blank
replace ofc with !(amoff) , mdl with !(cmdl)
replace modelpoc with !(ampoc)
erase
endif
enddo while flag
** program: dmain.prg
*** purpose: to delete maintainers(offices) for an existing model
** programmer: al noel
** files used: maintains
*
set up loop to do work
store t to flag
do while flag
erase
?
? 'Enter symbol for office no longer maintaining this Model and RETURN'
accept 'To quit deleting maintainers hit space bar and RETURN' to amoff
* check to quit or continue work
if amoff = ' ' .or. amoff = ' ' .or. amoff = ' '
store f to flag
use maintains
* actually remove records marked for deletion
pack
erase
return
else
erase

```

```

14,3 say 'Standby please... '
use maintains
mark for deletion records cf model edited and for office given by user
do while .not. eof
if mdl = cmdl .and. ofc = amoff
delete
endif
endif skip
enddo
endif
enddo while flag

** Program CHG-ELEMENT
** purpose: to permit user to change record about an existing element
** programmer: al noel
** files used: element, ielement, contains
** programs called: aefil, defil

set exact on
SET FORMAT TO SCREEN
SET COLOR TO 30,14
SET TALK OFF
ERASE
REMARK STANDBY PLEASE...
USE ELEMENT INDEX IELEMENT
* set up loop to do work
STORE t TO cmmore
DO WHILE CMMORE
ERASE
?
?
accept Enter the unique name of the Element you wish to change to CMNUM
ERASE
* locate record for unique name given by user
FIND &CMNUM
* if no find, tell user and ask if he wishes to continue
IF = 0 ERASE
?
?
? A Record with Element named CMNUM, is not in the file
ACCEPT Enter a Y to try again or N if not and then hit RETURN to cmansw
* check if wishes to quit or continue

```

```

IF CMANSW <> Y .AND. CMANSW <> Y
  STORE F TO CMMORE
ENDIF
ELSE
  * store file data into memory variables so user can edit it
  store elname to oldelname
  store elname to amelname
  store version to amversion
  store date to amdate
  store segnum to amsegnum
  store size to amsize
  store poc to ampoc
  store purpose to ampurpose
  store descrip to amdescrip
  * set up screen
  set format to eldat

READ * replace file data with edited data from screen
      REPLACE elname with !(amelname), version with !(amversion)
      REPLACE date WITH !(amdate), segnum WITH !(amsegnum)
      REPLACE size WITH !(amsize), poc WITH !(ampoc)
      replace
      * if unique name changed, then make changes in contains file
      if !(amelname) <> oldelname
        14,3 SAY 'STANDBY PLEASE...'
        use contains
        do while .not. eof
          if ename = cldelname
            replace ename with !(amelname)
            .and. etype = 'ELEMENT'
          endif
          skip
        enddo
        use contains
        do while .not. eof
          if trim(ename1) = TRIM(oldelname) .and. trim(etype1) = 'ELEMENT'
            replace ename1 with !(amelname)
          endif
          skip
        enddo
      endif
    ERASE ? Do you need to add new files that contain this element ?
    * check if user needs to new files containing the element
    accept If so, hit Y and RETURN, else hit space bar and RETURN to answer
    if 'Y' = !(answer)
      ERASE

```

```

do aeofil
endif
? ?
ERASE 'Delete file which no longer contain this element ?, Y or space' to ans
* check if user needs to delete files from containers that no longer contain
* element
if 'Y' = '(ans )
erase
do defil
endif ERASE
? ?
? ?
* prompt user for choice to change another record or to leave pgm
ACCEPT ENTER A Y IF YOU WANT TO CHANGE ANOTHER ELEMENT TO CMCONT
ERASE
IF CMCONT <> Y .AND. CMCONT <> Y
ERASE
STORE F TO CMMORE
ELSE
USE ELEMENT INDEX IELEMENT
ENDIF
ENDIF
*END OF DO WHILE
ENDDO
ERASE
RELEASE ALL
RETURN

```

```

* program: aeofil.prg
* *
* purpose: to add files (containers) for elements that preexist
* *
* programmer: al noel
* *
* files used: contains
* *
* set up loop to do work
store t to offlag

```

```

do while offlag
  ERASE
  ??
  ??
  ?? Enter the name for the file containing this element and RETURN
  accept To quit entering files hit the space bar and RETURN to amoff
  * check if user wants to quit or continue
  if amoff = '.or. amoff = ' '
    store f to offlag
    erase
    return
  else
    * create new record and load with file given and element changed
    use contains
    append blank
    replace ename1 with !(amoff), etype1 with FILE, ename with !(ame1name)
    replace etype with ELEMENT
  endif
enddo WHILE OFFLAG
** program: defil.prg
*** purpose: to delete files from contains file that no longer contain an element
*** programmer: al noel
*** files used: contains
store t to flag
do while flag
  erase
  ??
  ?? Enter name for file no longer containing this element and RETURN
  accept To quit deleting files hit space bar and RETURN to amoff
  * check if user wants to quit or continue
  if amoff = ' ' or. amoff = ' '
    store f to flag
    use contains
    * actually remove records marked for deletion
    pack
    erase
    return
  else
    erase
    14,3 say 'Standby please...'
    * mark for deletion records with file given and element changed
    use contains
    do while .not. eof
      if ename = ame1name .and. etype = 'ELEMENT'.and. ename1 =!(amoff)

```



```

        delete
      endif
      skip
    enddo
  endif
enddo while flag

* Program CHG-CATEGORY.PRG
** purpose: to change an existing category record
** programmer: al noel
** files used: category, icategory
**
SET FORMAT TO SCREEN
* set color to yellow and blue
SET COLOR TO 30,14
SET TALK OFF
ERASE
REMARK STANDBY PLEASE ...
USE CATEGORY INDEX ICATEGORY
* set up loop to do work
STORE T TO COMORE
DO WHILE COMORE
  ERASE
  ?
  ?
  ACCEPT ENTER THE UNIQUE NAME OF THE RECORD TO BE CHANGED TO PART1
  ACCEPT ENTER THE TYPE OF ENTITY TO PART2
  STORE PART1 - PART2 TO KEY
  ERASE
  * locate record with unique names given by user
  FIND &KEY
  IF
    * if record cant be located, tell user
    = 0
  ELSE
    ?
    ?
    ? A RECORD WITH NAME PART1 OF TYPE: PART2, 'NOT IN THE FILE'
    ACCEPT ENTER A Y IF YOU WISH TO TRY AGAIN TO COANSW
    IF COANSW <> Y .AND. COANSW <> Y
      store f to comore
    endif
  else
    * store file data in memory variable
    STORE ENAME TO AMENAME
    STORE ETYPE TO AMETYPE

```

```

STORE CAT TO AMCAT
* set up screen for user to use
SET FORMAT TO CATDAT

READ * replace data in file with edited data from screen
REPLACE ENAME WITH !(AMENAME), ETYPE WITH !(AMETYPE)
REPLACE CAT WITH !(AMCAT)
ERASE
?
ACCEPT ENTER A Y IF YOU WANT TO CHANGE ANOTHER RECORD TO COCONT
ERASE
IF COCONT <> Y .AND. COCONT <> Y
ERASE
STORE F TO COMORE
ENDIF
ENDIF
ENDDO
ERASE
RELEASE ALL
RETURN

```

```

* Program CHG-REPORT
* * purpose: to change record about an existing report
* * programmer: al noel
* * files used: report, ireport, outputs, uses
* * programs called; aruse, druse, arout, drout
* * set exact on
SET FORMAT TO SCREEN
SET COLOR TO 30,14
SET TALK OFF
ERASE
USE REPORT INDEX IREPORT
* set up loop to do work
STORE t TO CMmore
DO WHILE CMmore

```

```

ERASE
??
accept Enter the unique name of the Report you wish to change TO CMNUM
* locate record for unique name given by user
FIND %CMNUM
* if no find, tell user and ask if wants to quit
IF = 0
  ERASE
  ??
  ? A Record with Report named , CMNUM, is not in the file
  ACCEPT Enter a Y to try again or N if not and then hit RETURN to cmansw
  * check if user wants to quit
  IF CMANSW <> Y .AND. CMANSW <> Y
    STORE F TO CMMORE
  ENDIF
ELSE
  * store file data into memory variables for screen editing
  store rname to oldrname
  store fulname to cfullname
  store purpose to cpurpose
  store descrip to cdescrip
  store number to cnumber
  store status to cstatus
  store seclclass to cseclclass
  store outformat to coutformat
  store period to cperiod
  * put data up on screen for user to edit it
  2,1 SAY Unique name of Report get crname
  4,1 SAY Full name of Report get cfullname
  6,1 SAY Purpose get cpurpose
  8,1 SAY Description get cdescrip
  10,1 SAY Number get cnumber
  12,1 say Status get cstatus
  14,1 say Security Classification get cseclclass
  16,1 say Output Format get coutformat
  18,1 say Periodicity get cperiod
  READ * replace data in file with edited data from screen ! (cfullname)
  REPLACE rname WITH ! (crname) fulname WITH ! (cfullname)
  REPLACE purpose WITH ! (cpurpose) , descrip WITH ! (cdescrip)
  REPLACE number WITH ! (cnumber) , status WITH ! (cstatus )
  replace seclclass with ! (cseclclass ) , period with ! (cperiod )

```

```

* if unique name changed, change uses and outputs files accordingly
if crname <> oldrname
  ERASE
  14,3 SAY 'STANDBY PLEASE...'
  use uses
  do while .not. eof
    if ename = oldrname .and. etype = 'REPORT'
      replace ename with !(crname)
    endif
    skip
  enddo
  use outputs
  do while .not. eof
    if ename = oldrname .and. etype = 'REPORT'
      replace ename with !(crname)
    endif
    skip
  enddo
endif
ERASE
? Do you need to add new users of this Report ?
* check if new uses records to be added to uses file
accept 'Do you wish to delete any users, Y or N' to answer
if 'Y' = !(answer)
  ERASE
  do aruse
  endif
endif
?
?
erase
* check to delete users (offices) of the report, edited
accept 'Do you wish to delete any users, Y or N' to answer
if 'Y' = !(answer)
  erase
  do druse
endif
erase
?
?
?
? Do you wish to add a Model that outputs this report ? '
* check to see if new models output the report
accept 'If so enter a Y and RETURN, else hit the space bar and RETURN' to answer
if 'Y' = !(answer)
  erase
  dc arout
endif
erase
?

```

```

? * check to delete models from outputs file that used to output file
? * 'Do you wish to delete a Model that outputs this report?'
accept 'If so enter a Y and RETURN, else hit the space bar and RETURN' to answer
if 'Y' = '!' (answer)
    erase
    do drouit
endif
    ERASE
    ?
    ?
    ?
* prompt user for choice to change another record or to leave pgm
ACCEPT ENTER A Y IF YOU WANT TO CHANGE ANOTHER REPORT TO CMCONT
ERASE
IF CMCONT <> Y .AND. CMCCNT <> Y
    ERASE
    STORE F TO CMMORE
ELSE
    USE REPORT INDEX IREPORT
ENDIF
ENDIF
*END OF DO WHILE
ENDDO
ERASE
RELEASE ALL
RETURN
140

```

```

* program adreuse.prg
* * * purpose: to add users (offices) in uses file for reports that preexist
* * * programmer: al noel
* * * files used: uses
store t to offlag
do while offlag
    ERASE
    ?
    ?
    ? Enter the symbol for the office using this report and RETURN

```

```

accept To quit entering offices hit the space bar and RETURN to amoff
* check if user wants to quit or not
if amoff = ' ' .or. amoff = ' ' .or. amoff = ' '
  store f to offlag
  erase
  return
else
  use uses
  * put in new record
  append blank
  replace ofc with !(amoff), ename with !(cname), etype with 'REPORT'
  ENDDO
  ** program: druse.prg
  ** purpose: to delete users of a report
  ** programmer: al noel
  ** files used: uses
  store t to flag
  * set up loop to do work
  do while flag
    erase
  ?
  ? 'Enter symbol for office no longer using this Report and RETURN'
  accept 'To quit deleting users hit space bar and RETURN' to amoff
  * check if user wants to quit or not
  if amoff = ' ' .or. amoff = ' ' .or. amoff = ' '
    store f to flag
    use uses
    * actually remove records marked for deletion
    pack
    erase
    return
  else
    erase
    14,3 say 'Standby please...'
    * mark records for deletion for report edited and office given by user
    use uses
    do while .not. eof
      if ename = cname .and. ofc = amoff
        delete
      endif
    enddo
    skip
  endif
enddo while flag

```



```

** program arout.prg to add model that outputs a preexisting report
** purpose: to add model to outputs file for a preexisting report
** programmer: al noel
** files used: outputs
** ?
** ?
accept 'Enter unique name of Model and hit RETURN' to mod
*create new blank record and load with report edited and model given by user
use outputs
append blank
replace ename1 with !(mod) etype1 with 'MODEL'
replace ename with !(cname), etype with 'REPORT'
erase
return

** program: drout.prg
** purpose: to delete models in output file that produce a preexisting report
** programmer: al noel
** files used: outputs
erase
** ?
** ?
accept 'Enter the unique name of the Model and hit RETURN' to mod
erase
14,3 say 'Standby ...'
use outputs
store !(cname) to cname
store !(mod) to mod
* mark records for report edited and model given by user
do while .not. eof
if ename1 = mod .and. etype1 = 'MODEL'.and. ename .and. etype = 'REPORT'
delete
endif
skip
enddo
use outputs
* actually remove records marked for deletion
pack
return
** Program CHG-OFFICE
**

```

```

* purpose: to permit user to change an office record
** programmer: al noel
** files used: office, ioffice, uses , maintains
*
set exact on
set format to screen
** set color to yellow and blue
set color to 30,14
set talk off
erase
remark standby please...
use office index ioffice
store t to comore
do while comore
  erase
  ?
  ?
  ?
  accept enter the office symbol of the record to be changed to key
  erase
  * find record in file based upon what user gave just above
  find &key
  * tell user there was no finding of record for symbol he gave
  if
    = 0
    erase
    ?
    ?
    ?
    ?
    a record with the symbcl key, not in the file
    accept enter a y if you wish to, try again to coansw
    * check to quit or try again
    if coansw <> y .and. coansw <> y
      store f to comore
    endif
  else
    * store field variables to memory variable for editing
    store ofc to oldofc
    store ofc to cofc
    store oname to coname
    store bldg to cblbg
    store room to croom
    store phone to cphone
    store comments to ccomments
    store poc to cpoc
  * put up screen for user to edit data

```

```

1,1 SAY OFFICE SYMBOL GET COFC
3,1 SAY OFFICE NAME GET CONAME
5,1 SAY BUILDING GET CBLDG
7,1 SAY ROOM NUMBER GET CROOM
9,1 SAY PHONE GET CPHONE
11,1 SAY COMMENTS GET CCOMMENTS
13,1 SAY POINT OF CONTACT GET CPOC

READ * replace data in file record with data from screen
      REPLACE ONAME WITH ! (CONAME) , OFC WITH ! (COFC)
      REPLACE BLDG WITH ! (CBLDG) , ROOM WITH ! (CROOM) , PHONE WITH ! (CPHONE )
      REPLACE COMMENTS WITH ! (CCOMMENTS) , POC WITH ! (CPOC )
      * if new unique name for office then change associate intersection records
      * in uses and maintains files
      IF ! (COFC) <> OLDOFC
        ERASE
        14,7 SAY 'STAND BY PLEASE ...'
        USES
        DO WHILE .NOT. EOF
          IF TRIM(OFC) = TRIM(OLDOFC)
            REPLACE OFC WITH ! (COFC)
          ENDIF
          SKIP
        ENDDO
        USE MAINTAINS
        DO WHILE .NOT. EOF
          IF TRIM(OFC) = TRIM(OLDOFC)
            REPLACE OFC WITH ! (COFC)
          ENDIF
          SKIP
        ENDDO
      ENDIF
      ERASE
      ? ?
      * check if user wants to change another record
      ACCEPT ENTER A Y IF YOU WANT TO CHANGE ANOTHER RECORD TO COCONT
      ERASE
      IF COCONT <> Y .AND. COCONT <> Y
        ERASE
        STORE F TO COMORE
      ELSE
        USE OFFICE INDEX IOFFICE
      ENDIF
    ENDIF
  ENDDC
  ERASE

```

```

CLEAR
RELEASE ALL
RETURN

** Program CHG-ALIAS
** purpose: to change an existing alias record
** programmer: al noel
** files used; alias, ialias
set exact on
SET FORMAT TO SCREEN
** set color to yellow and blue
SET COLOR TO 30,14
SET TALK OFF
ERASE
REMARK STANDBY PLEASE...
USE ALIAS INDEX IALIAS
** set up loop to do work
STORE T TO COMORE
DO WHILE COMORE
  ERASE
  ? ? ? ?
  * BUILD SEARCH KEY AND THEN GO FIND RECORD YOU WANT
  ACCEPT ENTER THE UNIQUE NAME OF THE RECORD TO BE CHANGED TO PART1
  ACCEPT ENTER THE ALIAS OF THE ENTITY TO PART2
  STORE PART1 - PART2 TO KEY
  ERASE
  * find the unique record the user wants, a concatenated string
  FIND &KEY
  *if record can't be found, tell user
  IF
  = 0
  ERASE
  ? ? ? ?
  ? A RECORD WITH NAME PART1, OF TYPE: PART2, 'NOT IN THE FILE'
  ACCEPT ENTER A Y IF YOU WISH TO TRY AGAIN TO COANSW

```

```

* check if user wants to try again
IF COANSW <> Y.AND. COANSW <> Y
  store f to comore
endif
else
  * store file data into memory variables
  STORE ENAME TO AMENAME
  STORE ANAME TO AMANAME
  STORE ETYPE TO AMETYPE
  * put up screen so user can edit data
  SET FORMAT TO ALDAT
READ
* replace data in record with data from screen user has edited
  REPLACE ENAME WITH !(AMENAME) ANAME WITH !(AMANAME)
  REPLACE ETYPE WITH !(AMETYPE)
  ERASE
  ?
  ?
  ?
  ACCEPT ENTER A Y IF YOU WANT TO CHANGE ANOTHER RECORD TO COCONT
  ERASE
  * check if user wants to change another record
  IF COCONT <> Y.AND. COCONT <> Y
    ERASE
    STORE F TO COMORE
  endif
endif
  ENDDO
  ERASE
  RELEASE ALL
  RETURN

* Program CHG-PROGRAM
* * purpose: to permit user to change a program record in the dictionary
* * programmer: al noel
* * files used: program, iprogram, calls
* * programs called: apmod, dpmo
  set exact on
  SET FORMAT TO SCREEN
  SET COLOR TO 30,14
  SET TALK OFF
  ERASE

```

```

REMARK STANDBY PLEASE...
USE PROGRAM INDEX IPROGRAM
* set up loop to do work
STORE t TO CMORE
DO WHILE CMORE
  ERASE
  ?
  ?
  ? Enter the unique name of the Program you wish to change
  accept Please string together Program name and version, i.e. UTRACS1 TO CMNUM
  ERASE
  * look for record with unique name given by user
  FIND &CMNUM
  * if don't find record tell user and ask if wishes to try again
  IF =
    0 ERASE
    ?
    ?
    ? A Record with Program named CMNUM, is not in the file
    ACCEPT Enter a Y to try again or N if not and then hit RETURN to CMANSW
    * check if user wants to try again
    IF CMANSW <> Y -AND- CMANSW <> Y
      STORE F TO CMORE
    ENDIF
  ELSE
    * store data from file into memory variables for screen and editing
    store version to oldversion
    store pname to oldpname
    store pname to ampname
    store location to amlocation
    store language to amlanguage
    store version to amversion
    store purpose to ampurpose
    store descrip to amdescrip
    store status to amstatus
    store period to amperiod
    * put up screen for user to edit data
    set format to prodat
  READ
  * replace data in record with data from screen
  REPLACE pname WITH !(ampname), location with !(amlocation)
  REPLACE version WITH !(amversion), purpose with !(ampurpose)
  REPLACE descrip WITH !(amdescrip), status WITH !(amstatus)
  replace
  * if unique name has been changed, go change intersection records
  if (ampname <> oldpname) .or. (amversion <> oldversion)

```



```

ERASE
14,3 'STANDBY PLEASE...'
use calls
do while .not. eof
  if ename = (oldpname-oldversion) .and. etype = 'PROGRAM'
    store ampname-amversion to holder
    replace ename with !(holder)
  endif
  skip
enddo

endif
erase
* Check to add intersection records
? Do you need to add new models that call this program?
accept If so, hit Y and RETURN, else hit space bar and RETURN to answer
if 'Y' = !(answer)
  ERASE
  do apmod
  endif
endif

? ?
erase
* Check to delete models from intersection file
accept 'Delete models which no longer call this program ?, Y or space' to ans
if 'Y' = !(ans)
  erase
  do dpmod
  endif
endif
ERASE
? ? ?
* prompt user for choice to change another record or to leave pgm
ACCEPT ENTER A Y IF YOU WANT TO CHANGE ANOTHER PROGRAM TO CMCONT
ERASE
IF CMCONT <> Y .AND. CMCONT <> Y
  ERASE
  STORE F TO CMMORE
ELSE
  USE PROGRAM INDEX I PROGRAM
ENDIF
ENDIF

*END CF DO WHILE
ENDDO
ERASE
RELEASE ALL
RETURN

```

```

** apmod.prg      to add models in calls file for preexisting program records
** purpose: to add models in calls file for preexisting program records
** programmer: al noel
** files used: calls
store t to offlag
do while offlag
  ERASE
  ??
  ??
  ?? Enter the unique name for the model calling this program and RETURN
  accept to quit entering models hit the space bar and RETURN to amoff
  * check to quit
  if amoff = '.or.' or amoff = ' ' .or. amoff = ' '
    store f to offlag
    erase
    return
  else
    * create new blank record and load data from screen
    use calls
    append blank
    store ampname - amversion to holder
    replace ename1 with !(amoff), etype1 with 'MODEL'
    replace ename with !(holder); etype with 'PROGRAM'
  endif
  ENDDO WHILE OFFLAG
** dpmc.d.prg
** purpose: to delete intersection records from calls file that are for models
**           that no longer call the program being charged in chg-pro
** programmer: al noel
** files used: calls
store t to flag
do while flag
  erase
  ??

```

```

? 'Enter unique name for model no longer calling this program and RETURN'
accept 'To quit deleting models hit space bar and RETURN' to amoff
* check to quit
if amoff = ' ' .or. amoff = ' ' .or. amoff = ' '
  store f to flag
  use calls
  pack
  erase
  return
else
  erase
  14,3 say 'Standby please...'
  * delete record from calls file for the program edited & model given by user
  do while .not. eof
    STORE AMPNAME-AMVERSION TO X
    if TRIM(ename1) = TRIM(amoff) .and. etype1= 'MODEL'.and. ;
      TRIM(ENAME) = TRIM(X)
      delete
    endif
  enddo
  skip
endif
enddo while flag

```

# Format Files Used in CHANGE subsystem

## CATDAT.FMT

```

2,3 say Entity's unique name get amename
4,3 say Type get ametype
6,3 say Category get amcat

```

## PRODAT.FMT

```

2,3 say Program's unique name get ampname
4,3 say Location get amlocation
6,3 say Version get amversion
8,3 say Status get amstatus
10,3 say Purpose get ampurpose
12,3 say Description get amdescrip
14,3 say Periodicity get amperiod
16,3 say Language get amlanguage

```

## ALDAT.FMT

```

2,3 say Entity's unique name get amename

```

```

4,3 say      Alias get amaname
6,3 say      Type of entity (i.e. MODEL, or REPORT) get ametype

FEDAT.FMT

2,3 say      Field's unique name get amfld
4,3 say      Length of field in bytes get amlength
6,3 say      Data type (A = alpha, N = numeric, S = special) get amdatatype
8,3 say      Null status? (Y or N) get amnullstat
10,3 say     Source get amsource
12,3 say     Domain get amdomain
14,3 say     Purpose get ampurpose
16,3 say     Description get amdescrip
18,3 say     Status get amstatus
20,3 say     Security classification (U,C,S,TS) get amseclclass
22,3 say     Point of contact get am poc

```

#### FIDAT.FMT

```

2,3 say      File's unique name get amfname
4,3 say      File's full name get amfullname
6,3 say      Storage media get ammedia
8,3 say      Status get amstatus
10,3 say     Purpose get ampurpose
12,3 say     Description get amdescrip
14,3 say     Point of contact get am poc
16,3 say     Security classification get amseclclass
18,3 say     Date of last update get amlstupdat

```

#### ELDAT.FMT

```

2,3 say      Element's unique name get amelname
4,3 say      Version get amversion
6,3 say      Date get amdate
8,3 say      Sequence number get amseqnum
10,3 say     Size get amsize
12,3 say     Point of Contact get am poc
14,3 say     Purpose get ampurpose
16,3 say     Description get amdescrip

```

```

** program name: del-menu
** purpose: to permit user to delete records from dictionary
** programmer: al noel
**

```

```
set color to 30,14
SET TALK OFF
```

```
ERASE
*set up loop for user to select specific function
DO WHILE T
2,0
```

```
TEXT
```

This is the DELETE menu. It permits you to delete records from any file. Please enter the number of the function you wish to perform.

0 to quit

- |                       |                      |
|-----------------------|----------------------|
| 1 Model (DEL-MODEL)   | 5 File (DEL-FI)      |
| 2 Office (DEL-OFFICE) | 6 Element (DEL-EL)   |
| 3 Report (DEL-REP)    | 7 Field (DEL-FE)     |
| 4 Program (DEL-PRO)   | 8 Alias (DEL-AL)     |
|                       | 9 Category (DEL-CAT) |

Enter the selection here

```
ENDTEXT
```

```
WAIT TO DMSELECT
```

```
20,0 SAY
```

```
21,0 SAY
```

\* case statement for calling of program per users choice above

```
DO CASE
CASE DMSELECT = 0
```

```
USE
```

```
RELEASE ALL
```

```
ERASE
```

```
RETURN
```

```
CASE DMSELECT = 1
```

```
DO DEL-MODEL
```

```
CASE DMSELECT = 2
```

```
DO DEL-OFFICE
```

```
CASE DMSELECT = 3
```

```
DO DEL-REP
```

```
CASE DMSELECT = 4
```

```
DO DEL-PRO
```

```
CASE DMSELECT = 5
```

```
DO DEL-FI
```

```
CASE DMSELECT = 6
```

```
DO DEL-EL
```

```
CASE DMSELECT = 7
```

```
DO DEL-FE
```

```
CASE DMSELECT = 8
```

```
DO DEL-AL
```

```
CASE DMSELECT = 9
```

```
DO DEL-CAT
```

```

* error message if wrong number picked
OTHERWISE
  ERASE
  20,3 SAY Please enter values between 0 and 9 only
  21,3 SAY Please try again
ENDCASE
ENDDO
RETURN

* Program: name del-file
** purpose: to delete file record from dictionary
** programmer: al noel
** files used: file, ifile, outputs, inputs, contains
set exact on
SET FORMAT TO SCREEN
ERASE
SET COLOR TO 30,14
SET TALK OFF
* store t to dmmore
* This starts a loop which runs the program until user wants to quit
do while dmmore
  erase
  use file      index ifile
  ??
ACCEPT ENTER THE FILE'S    UNIQUE NAME PLEASE TO DMNUM
ERASE
* This locates the exact record
  IF FIND &DMNUM
    = 0
    ??
    ??
    ??
    ? A FILE NAMED:      DMNUM, IS NOT IN THE FILE, SORRY
    ACCEPT ENTER A Y IF YOU WISH TO TRY AGAIN TO DMANSW
    * This allows the user to make a mistake
    ERASE
    IF !(DMANSW) <> Y
      IF STORE F TO DMMORE
        ENDDIF
      ELSE

```



```

store fname      to dmmodname
store purpose    to dmmodlnum
7,1 say THE FILE'S UNIQUE NAME IS:
7,30 SAY DMMODNAME
9,1 say THE FILE'S PURPOSE IS:
9,30 SAY DMMODELNUM

***** WARNING! ***** WARNING! ***** WARNING! *****
* This allows the user to insure they wish to proceed.
* ACCEPT IF YOU ARE SURE THIS IS THE RECORD TO BE DELETED ENTER A Y TO PROCEED
* The delete command only marks the record for deletion.
if !(PROCEED) = Y
  ERASE
  14,8 SAY 'STANDBY PLEASE...'
  DELETE
  USE OUTPUTS
  DELETE ALL FOR TRIM(ENAME) = TRIM(DMNUM) .AND. TRIM(ETYPE) = 'FILE'
  USE INPUTS
  DELETE ALL FOR TRIM(ENAME) = TRIM(DMNUM) .AND. TRIM(ETYPE) = 'FILE'
  USE CONTAINS
  DELETE ALL FOR TRIM(ENAME1) = TRIM(DMNUM) .AND. TRIM(ETYPE1) = 'FILE'
ENDIF
ERASE
?
?
* This allows the user to continue deleting records.
* ACCEPT DO YOU WISH TO CONTINUE DELETING RECORDS? IF SO ENTER A Y TO DOAGI
IF DOAGI <> Y .AND. DOAGI <> Y
  ERASE
  STORE F TO DMMORE
ENDIF
ENDIF
ENDDO
* The program is exited.
ERASE
REMARK STANDBY WHILE I PERFORM MAINTENANCE ON ASSOCIATED FILE...
* actually delete records marked for deletion above
USE OUTPUTS
PACK
USE INPUTS
PACK
USE FILE
PACK
USE CONTAINS
PACK
ERASE

```

RELEASE ALL  
RETURN

```

* Program: name del-office
* * purpose: to delete office records from dictionary
* * programmer: al noel
* * files used: office, ioffice, uses, maintains
*
set exact on
SET FORMAT TO SCREEN
SET TALK OFF
SET COLOR TO 30,14
ERASE 12,3 SAY STANDBY PLEASE...
use office index ioffice
store t to domore
* This starts a loop which runs the program until the user wants to quit
do while domore
erase
?
?
ACCEPT ENTER THE OFFICE SYMBOL CF THE RECORD TO BE DELETED TO X
STORE ! (x) TO KEY
ERASE
FIND &KEY
IF
= 0
ERASE
?
?
? AN OFFICE WITH THE SYMBOL OF : KEY, NOT IN FILE
ACCEPT ENTER A Y IF YOU WISH TO TRY AGAIN TO DOANSW
* This allows the user to make a mistake
ERASE
IF DOANSW <> Y .AND. DOANSW <> Y
STORE F TO DOMORE
ENDIF
ELSE
*put data up so user can be sure he has the correct OFFICE to delete
store OFC TO DOSYM
store ONAME TO DOROLE
*
7,1 say OFFICE SYMBOL IS

```

```

7,18 SAY DOSYM
9,1 SAY OFFICE NAME IS
9,19 SAY DOROLE
? ?
? ?
***** WARNING! ***** WARNING! ***** WARNING! *****
ACCEPT IF YOU ARE SURE THIS IS THE RECORD TO BE DELETED ENTER A Y TO PROCEED
* The delete command only marks the record for deletion
IF ! (PROCEED) = Y
  ERASE
  14,7 SAY ' STANDBY PLEASE...'
  DELETE
  USE USES
  DELETE ALL FOR TRIM(OFC) = TRIM(KEY)
  USE MAINTAINS
  DELETE ALL FOR TRIM(OFC) = TRIM(KEY)
ENDIF
ERASE
? ?
* This allows the user to continue deleting records? IF SO ENTER A Y TO DOAGI
ACCEPT DO YOU WISH TO CONTINUE DELETING RECORDS?
IF DOAGI <> Y .AND DOAGI <> Y
  ERASE
  STORE F TO DOMORE
ELSE
  USE OFFICE INDEX IOFFICE
ENDIF
ENDIF
ERASE
ENDDO
REMARK STANDBY PLEASE
* The program is exited
  USE OFFICE
* The pack command actually does the deletion
  PACK USES
  PACK
  USE MAINTAINS
  PACK
  ERASE
  RELEASE ALL
  RETURN
*****
* Program: name del-category
**

```

```

* purpose: to delete category records from dictionary
** programmer: al noel
** files used: category, icategory
set exact on
SET FORMAT TO SCREEN
SET TALK OFF
SET COLOR TO 30,14
ERASE t to domore
** This starts a loop which runs the program until the user wants to quit
do while domore
    use category index icategory
    erase
    ?
    ?
ACCEPT ENTER THE UNIQUE NAME OF THE RECORD TO BE DELETED TO PART1
ACCEPT ENTER THE TYPE OF ENTITY TO PART2
STORE PART1 - PART2 TC KEY
ERASE
FIND &KEY
IF
    = 0
ERASE
    ?
    ?
    ? AN ENTITY WITH THE NAME OF : KEY AND TYPE: PART2, NOT IN FILE
    ACCEPT ENTER A Y IF YOU WISH TO TRY AGAIN TO DOANSW
    * This allows the user to make a mistake
    ERASE
    IF DOANSW <> Y .AND. DOANSW <> Y
        STORE F TO DOMORE
    ENDIF
ELSE
    *put data up so user can be sure he has the correct OFFICE to delete
    store ENAME TO DOSYM
    store ETYPE TO DOROLE
    *
    7,1 say THE ENTITY'S NAME IS:
    7,23 SAY DOSYM
    9,1 SAY THE ENTITY'S TYPE IS:
    9,23 SAY DOROLE
    ?
    ?
    ***** WARNING! ***** WARNING! ***** WARNING! *****

```

```

ACCEPT IF YOU ARE SURE THIS IS THE RECORD TO BE DELETED ENTER A Y TO PROCEED
* The delete command only marks the record for deletion
IF !(PROCEED) = Y
  DELETE
ENDIF
ERASE
??
?
* This allows the user to continue deleting records
ACCEPT DO YOU WISH TO CONTINUE DELETING RECORDS? IF SO ENTER A Y TO DOAGI
IF DOAGI <> Y .AND DOAGI <> Y
  ERASE
  STORE F TO DOMORE
ENDIF
ENDIF
ERASE
ENDDO
ERASE
PEMARK STANDBY WHILE I PERFORM MAINTENANCE ON THE FILE
* actually remove records marked for deletion above
USE CATEGORY
PACK
ERASE
RELEASE ALL
RETURN

* Program: name del-field
* * purpose: to delete field records from dictionary
* * programmer: al noel
* * files used: field, ifield, contains
*
set exact on
SET FORMAT TO SCREEN
ERASE
SET COLOR TO 30,14
SET TALK OFF
store t to dmore
* This starts a loop which runs the program until user wants to quit
do while dmore
  erase
  use field index ifield
?

```

```

? ACCEPT ENTER THE FIELD'S UNIQUE NAME PLEASE TO DMNUM
ERASE
* This locates the exact record
IF FIND &DMNUM
= 0
? ? ?
? ? A FIELD NAMED: DMNUM, IS NOT IN THE FILE, SORRY
? ACCEPT ENTER A Y IF YOU WISH TO TRY AGAIN TO DMANSW
* This allows the user to make a mistake
ERASE
IF !(DMANSW) <> Y
STORE F TO DMMORE
ENDIF
ELSE
store fld to dmmodname
store purpose to dmmodelnum
7,1 say THE FIELD'S UNIQUE NAME IS:
7,30 SAY DMMODNAME
7,6,1 SAY THE FIELD'S PURPOSE IS:
9,30 SAY DMMODELNUM
? ? ? ? ?
***** WARNING! ***** WARNING! *****
* This allows the user to insure they wish to proceed.
ACCEPT IF YOU ARE SURE THIS IS THE RECORD TO BE DELETED ENTER A Y TO PROCEED
* The delete command only marks the record for deletion.
if !(PROCEED) = Y
DELETE
USE CONTAINS
DELETE ALL FOR TRIM(ENAME) = TRIM(DMNUM) .AND. TRIM(ETYPE) = 'FIELD'
ENDIF
ERASE
? ? ?
* This allows the user to continue deleting records.
ACCEPT DO YOU WISH TO CONTINUE DELETING RECORDS? IF SO ENTER A Y TO DOAGI
IF DOAGI <> Y
ERASE
STORE F TO DMMORE
ENDIF
ENDIF
ENDDO
ERASE

```



```

REMARK STANDBY WHILE I PERFORM MAINTENANCE ON ASSOCIATED FILES
** actually remove records marked for deletion above
USE CONTAINS
PACK
  USE FIELD
PACK
  ERASE
RELEASE ALL
RETURN

* Program: name del-report
* purpose: to delete report records from dictionary
* programmer: al noel
* files used: report, ireport, uses, outputs
SET FORMAT TO SCREEN
set exact on
ERASE COLOR TO 30,14
SET TALK OFF
store t to dmmore
** This starts a loop which runs the program until user wants to quit
do while dmmore
  use report index ireport
  erase
  ?
  ?
accept ENTER THE REPORT'S UNIQUE NAME PLEASE TO DMNUM
ERASE
* This locates the exact record
  FIND &DMNUM
  IF 0
  =
  ?
  ?
  ?
  ?
  ? A REPORT NAMED:  DMNUM, IS NOT IN THE FILE, SORRY
  ACCEPT ENTER A Y IF YOU WISH TO TRY AGAIN TO DMANSW
  * This allows the user to make a mistake
  ERASE
  IF !(DMANSW) <> Y
  IF STORE F TO DMMORE
  ENDIF
ELSE

```

```

*put data up so user can be sure he has the correct model to delete
store rname to dmmodname
store purpose to dmmodelnum
7,1 say THE REPORT'S UNIQUE NAME IS:
7,30 SAY DMMODNAME
9,1 say THE REPORT'S PURPOSE IS:
9,30 SAY DMMODELNUM
? ? ? ? ?
***** WARNING! ***** WARNING! *****
? ? ? ? ?
* This allows the user to insure they wish to proceed.
ACCEPT IF YOU ARE SURE THIS IS THE RECORD TO BE DELETED ENTER A Y TO PROCEED
* The delete command only marks the record for deletion.
if !(PROCEED) = Y
DELETE
USE USES
DELETE ALL FOR TRIM(ENAME) = TRIM(DMNUM) .AND. TRIM(ETYPE) = 'REPORT'
USE OUTPUTS
DELETE ALL FOR TRIM(ENAME) = TRIM(DMNUM) .AND. TRIM(ETYPE) = 'REPORT'
ENDIF
ERASE
? ? ?
* This allows the user to continue deleting records.
ACCEPT DO YOU WISH TO CONTINUE DELETING RECORDS? IF SO ENTER A Y TO DOAGI
IF DOAGI <> Y .AND. DOAGI <> Y
ERASE
STORE F TO DMMORE
ENDIF
ENDIF
ENDDO
ERASE
REMARK STANDBY WHILE I PERFORM MAINTENANCE ON ASSOCIATED FILES
USE USES
PACK
USE OUTPUTS
PACK
USE REPORT
PACK
16,15 say GOOD BYE FROM THE DELETE REPORT PROGRAM
ERASE
RELEASE ALL
RETURN

```

```

* Program: name del-model
* purpose; to delete model records from dictionary
* programmer: al noel
* files used: model, imodel, inputs, outputs, uses, maintains
SET FORMAT TO SCREEN
set exact on
ERASE
SET COLOR TO 30,14
SET TALK OFF
*
store t to dmmore
* This starts a loop which runs the program until user wants to quit
do while dmmore
  USE MODEL INDEX IMODEL
  erase
  ?
  ?
accept ENTER MODEL'S UNIQUE NAME PLEASE TO DMNUM
ERASE
* This locates the exact record
  IF FIND &DMNUM
    = 0
    ?
    ?
    ?
    ?
    ? A MODEL NAMED:  DMNUM, IS NOT IN THE FILE, SORRY
    ACCEPT ENTER A Y IF YOU WISH TO TRY AGAIN TO DMANSW
    * This allows the user to make a mistake
    ERASE
    IF !(DMANSW) <> Y
      IF STORE F TO DMMORE
        ENDF
      ELSE
        *put data up so user can be sure he has the correct model to delete
        store mdi to dmmodname
        store modname to dmmodelnum
        7,1 say THE MODEL'S UNIQUE NAME IS:
        7,30 SAY DMODNAME
        9,1 SAY THE MODEL'S FULL NAME IS:
        9,30 SAY DMMODELNUM
        ?
        ?
        ***** WARNING! ***** WARNING! ***** WARNING! *****

```

```

?? * This allows the user to insure they wish to proceed.
* ACCEPT IF YOU ARE SURE THIS IS THE RECORD TO BE DELETED. ENTER A Y TO PROCEED
* The delete command only marks the record for deletion.
if !(PROCEED) = Y
  ERASE
  14 SAY 'STANDBY PLEASE...'
  DELETE
  USE USES
  DELETE ALL FOR TRIM(ENAME) = TRIM(DMNUM) .AND. TRIM(ETYPE) = 'MODEL'
  USE MAINTAINS
  DELETE ALL FOR TRIM(MDL) = TRIM(DMNUM)
  USE INPUTS
  DELETE ALL FOR TRIM(ENAME1) = TRIM(DMNUM) .AND. TRIM(ETYPE1) = 'MODEL'
  USE OUTPUTS
  DELETE ALL FOR TRIM(ENAME1) = TRIM(DMNUM) .AND. TRIM(ETYPE1) = 'MODEL'
  USE CALLS
  DELETE ALL FOR TRIM(ENAME1) = TRIM(DMNUM) .AND. TRIM(ETYPE1) = 'MODEL'
ENDIF
ERASE
?
* This allows the user to continue deleting records.
* ACCEPT DO YOU WISH TO CONTINUE DELETING RECORDS? IF SO ENTER A Y TO DOAGI
IF DOAGI <> Y .AND. DOAGI <> Y
  ERASE
  STORE F TO DMMORE
ENDIF
ENDIF
ENDDO
ERASE
REMARK STANDBY WHILE I PERFORM MAINTENANCE ON ASSOCIATED FILES...
* actually remove records marked for deletion above
USE USES
PACK
USE MAINTAINS
PACK
USE MODEL
PACK
USE OUTPUTS
PACK
USE INPUTS
PACK
USE CALLS
PACK
CLEAR
16, 15 say GOOD BYE FROM THE DELETE MODEL PROGRAM

```

```

ERASE
RELEASE ALL
RETURN

* Program: name del-program
* * purpose: to delete records for programs from dictionary
* * programmer: al noel
* * files used: program, iprogram, calls
SET FORMAT TO SCREEN
set exact on
ERASE
SET COLOR TO 30,14
SET TALK OFF
*
store t to dmmore
* This starts a loop which runs the program until user wants to quit
do while dmmore
  ERASE
  use program index iprogram
  ?
  ?
  ?
  ACCEPT 'ENTER THE PROGRAM'S UNIQUE NAME PLEASE'
  ERASE
  * PLEASE STRING TOGETHER PROGRAM NAME AND VERSION, I.E. UTRACS1 ' TO DMNUM
  * This locates the exact record
  FIND &DMNUM
  IF =
  ?
  ?
  ?
  ?
  ? A PROGRAM NAMED: , DMNUM, IS NOT IN THE FILE, SORRY
  ACCEPT ENTER A Y IF YOU WISH TO TRY AGAIN TO DMANSW
  * This allows the user to make a mistake
  ERASE
  IF !(DMANSW) <> Y
  IF STORE F TO DMMORE
  ENDIF
ELSE
ERASE
*put data up so user can be sure he has the correct model to delete
store pname to dmmodname
store version to dmversion

```

```

store purpose to dmmodelnum
7,1 say THE PROGRAM'S UNIQUE NAME IS:
7,30 SAY DMMODNAME
9,1 say THE VERSION NUMBER IS:
9,24 SAY DMVERSION
11,1 say THE PROGRAM'S PURPOSE IS:
11,30 SAY DMMODELNUM
? ?
? ?
? ?
***** WARNING! ***** WARNING! *****
* This allows the user to insure they wish to proceed.
ACCEPT IF YOU ARE SURE THIS IS THE RECORD TO BE DELETED. ENTER A Y TO PROCEED
* The delete command only marks the record for deletion.
IF !(PROCEED) = Y
  DELETE
  USE CALLS
  DELETE ALL FOR TRIM(ENAME) = TRIM(DMNUM) .AND. TRIM(ETYPE) = 'PROGRAM'
ENDIF
ERASE
? ?
* This allows the user to continue deleting records. IF SO ENTER A Y TO DOAGI
ACCEPT DO YOU WISH TO CONTINUE DELETING RECORDS? IF SO ENTER A Y TO DOAGI
IF DOAGI <> Y .AND. DOAGI <> Y
  ERASE
  STORE F TO DMMORE
ENDIF
ENDIF
ENDDO
ERASE
REMARK STANDBY WHILE I PERFORM MAINTENANCE ON ASSOCIATED FILES
*actually remove records marked for deletion above
USE CALLS
PACK
USE PROGRAM
PACK
ERASE
RELEASE ALL
RETURN

* Program: name del-alias
* *
* purpose: to delete alias records from dictionary
* *
* programmer: al noel
* *

```



```

* files used: alias, ialias
*
set exact on
SET FORMAT TO SCREEN
SET TALK OFF
SET COLOR TO 30,14
ERASE
store t to domore
* This starts a loop which runs the program until the user wants to quit
do while domore
    use alias index ialias
    erase
    ??
    ACCEPT ENTER THE UNIQUE NAME OF THE ENTITY TO BE DELETED TO PART1
    ACCEPT ENTER THE ENTITY'S ALIAS TO PART2
    STORE PART1 - PART2 TO KEY
    ERASE
    FIND &KEY
    IF
        = 0
    ERASE
    ??
    ? AN ENTITY WITH THE NAME OF : KEY, AND ALIAS OF: PART2, NOT IN FILE
    ACCEPT ENTER A Y IF YOU WISH TO TRY AGAIN TO DOANSW
    * This allows the user to make a mistake
    ERASE
    IF DOANSW <> Y .AND. DOANSW <> Y
        STORE F TO DOMORE
    ENDIF
    ELSE
    *put data up so user can be sure he has the correct OFFICE to delete
    store ENAME TO DOSYM
    store ANAME TO DOROLE
    *
    7,1 say THE ENTITY'S NAME IS:
    7,23 SAY DOSYM
    9,1 say THE ENTITY'S ALIAS IS:
    9, 23 SAY DOROLE
    ??
    ***** WARNING! ***** WARNING! ***** WARNING! *****
    * ACCEPT IF YOU ARE SURE THIS IS THE RECORD TO BE DELETED ENTER A Y TO PROCEED
    * The delete command only marks the record for deletion
    IF ! (PROCEED) = Y
        DELETE

```

```

ENDIF
ERASE
??
* This allows the user to continue deleting records
  ACCEPT DO YOU WISH TO CONTINUE DELETING RECORDS? IF SO ENTER A Y TO DOAGI
  IF DOAGI <> Y .AND DOAGI <> Y
    FRASE
    STORE F TO DOMORE
  ENDIF
ENDIF
ERASE
ENDDO
ERASE
REMARK STANDBY WHILE I PERFORM MAINTENANCE ON THE FILE
* actually remove records marked for deletion above
USE ALIAS
PACK
ERASE
RELEASE ALL
RETURN

* Program: name del-element
* * purpose: to delete an element record from the dictionary
* * programmer; al noel
* * files used: element, ielement, contains
SET FORMAT TO SCREEN
ERASE
SET COLOR TO 30,14
SET TALK OFF
set exact on
store t to dmmore
* This starts a loop which runs the program until user wants to quit
do while dmmore
  erase

```



```

ENDDO
ERASE
REMARK STANDBY WHILE I PERFORM MAINTENANCE ON ASSOCIATED FILES
USE CONTAINS
PACK
USE ELEMENT
PACK
ERASE
RELEASE ALL
RETURN

* PROGRAM: QERY-MENU
* * purpose: to permit user to select query program he desires
* * programmer: al noel
* * files used: model, office, report, file, element, field, uses, contains
* * maintains, inputs, outputs, calls, program
* * programs called: fortrev, baktrev, foral, bakal, forcat, bakcat
SET FORMAT TO SCREEN
set color to 30,14
set talk off
ERASE
* set up loop
DO WHILE T
1,0,TEXT
This is the QUERY menu. It permits you to have the dictionary process
ad-hoc queries. Queries are either of the 'forward' type where an entity
is the subject and you specify the verb and object or of the 'backward' type
where the entity you give is the object of an action and subject.

QUERY MENU

1) Forward type (i.e. P3M calls what programs ?) (FORTREV)
2) Backward type (i.e. P3M is used by what offices ?) (BAKTRREV)
3) Forward type for aliases ( indicate type and system provides all aliases)
(FORAL)
4) Backward type for aliases (give entity name and system provides all
aliases for that particular entity)
(BAKAL)
5) Forward type for categories (indicate type and system gives categories)
(FORCAT)
6) Backward type for categories (give entity name and system gives all
categories that entity is in)
(BAKCAT)

```

0 to quit

```
Enter the selection here
ENDTEXT
* wait for user choice of programs
WAIT TO RMSELECT
20,0 SAY
21,0 SAY
* set up case statement for calling of program per user choice
DO CASE RMSELECT = 0
USE
  clear
  ERASE
  RELEASE ALL
  RETURN
CASE RMSELECT = 1
  DO FORTREV
CASE RMSELECT = 2
  DO BAKTREV
CASE RMSELECT = 3
  DO foral
CASE RMSELECT = 4
  DO bakal
CASE RMSELECT = '5'
  DO forcat
CASE RMSELECT = '6'
  DO bakcat
OTHERWISE
  * error message for choice of number s not in range of case statement
  ERASE
  20,0 SAY Please enter values between 0 and 6 only
  21,0 SAY Please try again
ENDCASE
ENDDO
RETURN
```

```
** program: fortreve.prg
** purpose: for querying in a 'forward' manner, that is, subject then verb,
**           then object, i.e. model calls programs
** programmer: al noel
```

```

* files used: model, office, program, file, element, field, uses, maintains,
* *
* set exact on
SET FORMAT TO SCREEN
SET COLOR TO 30,14
set talk off
* set up loop
store t to loop
do while loop
erase
* init variables for screen
store , to git , to qs:office
store , to qv:uses
store , to go:reports
store , to qv:maintan
store , to go:model
store , to qs:model
store , to qv:calls
store , to go:program
store , to qv:inputs
store , to go:file
store , to qv:outputs , to qs:file
store , to qv:contain
store , to go:element
store , to qs:element
store , to go:fields
* put screen up for user to indicat subject and mark verb and object with x's
2,1 say , SUBJECTS
3,1 SAY , -----
5,3 SAY 'Office' get qs:office
5,30 say 'Uses' get qv:uses
5,52 say 'Reports' get go:reports
6,25 say 'Maintains' get qv:maintan
6,53 say 'Models' get go:model
9,29 say 'Calls' get qv:calls
9,29 say 'Programs' get go:program
10,28 say 'Inputs' get qv:inputs
11,27 say 'Outputs' get qv:outputs
10,54 SAY 'Files' get go:file
13,5 say 'File' get qs:file
14,26 say 'Contains' get qv:contain
13,51 say 'Element' get go:element
15,2 say 'Element' get qs:element
15,53 say 'Fields' get go:fields

```



```

17, 4 say 'To quit put an X in this field'          get git
remark Put the unique name of the entity you are querying about in the field
remark next to the entity's type. Then place an X in one appropriate verb
remark field for that entity type and an X in one appropriate object field.
remark I.E. Office:DAPC-PLF : Uses:X: Models:X:
remark To process query move cursor through remaining fields using RETURN key
read
store 'Strike any key to continue...' to msg
* if user wants to quit, then no blank in quit field
if quit <>
  erase
  clear
  release all
  return
else
  erase
  ** if subject not blank, then look for verb and use file associated with verb
  ** also load query message with appropriate subject and verb wordings
  ** subjects are marked as qs variables, verbs as gv and objects as go.
  ** display statements put up on screen, for conditions met
  ** loop commands jump around remaining code of loop to save checking
  ** if user mistakenly marks more than one subject, the first marked gets
  ** executed only then program jumps around remainder with loop statement
  ** program waits for use to hit any key before present in blank screen for
  ** another query
  ? if $(qs:office,1,2) <> ' '
  store 'The office '+qs:office to query
  if gv:uses <> ' '
    store query+'uses ' to query
    use uses
  else
    store query+'maintains ' to query
    use maintains
  endif
  if go:reports <> ' '
    store query+'the following reports ' to query
    ? query
    ?
    display ename for trim(qs:office) = trim(ofc) .and. trim(etype) ;
    = 'REPORT' OFF
    ?
    ? msg
    wait to continue
    loop
  endif
  if go:model <> ' ' .and. gv:uses <> ' '

```

```

store query+      ' the following models' to query
?
? query
?
display ename for trim(qs:office) = trim(ofc) .and. trim(etype) ='MODEL';
off
?
? msg
wait to continue
loop
endif
if go:model <> ' ' .and. gv:maintan <> ' '
store query+ 'the following models' to query
? query
?
display mdl for trim(qs:office) = trim(ofc) off
?
? msg
wait to continue
loop
endif
endif
if $(qs:model,1,2) <> ' '
store 'The model'+qs:model to query
if gv:calls <> ' '
store query+ 'calls the following programs ' to query
use calls
? query
?
display ename for trim(qs:model) = trim(ename1) .and. trim(etype1) = ;
'MODEL' .and. trim(etype) = 'PROGRAM' off
?
? msg
wait to continue
loop
endif
if gv:inputs <> ' '
store query+ ' inputs the following files ' to query
use inputs
? query
?
display ename for trim(qs:model) = trim(ename1) .and. trim(etype1) = ;
'MODEL' .and. trim(etype) = 'FILE' off
?
? msg
wait to continue
loop
endif

```

```

if gv:outputs <> ' '
store query+ ' outputs to the following files ' to query
use outputs
? query
?
display ename for trim(qs:model) = trim(ename1) .and. trim(etype1) = ;
'MODEL' .and. trim(etype) = 'FILE' OFF
?
? msg
wait to continue
loop
endif
if $(qs:file,1,2) <> ' '
store 'The file '+qs:file+ ' contains the following elements ' to query
use contains
? query
?
display ename for trim(qs:file) = trim(ename1) .and. trim(etype1) = ;
'FILE' .and. trim(etype) = 'ELEMENT' Off
?
? msg
wait to continue
loop
endif
if $(qs:element,1,2) <> ' '
store 'The element '+qs:element+ ' contains the following fields ' to query
use contains
? query
?
display ename for trim(qs:element) = trim(ename1) .and. trim(etype1) = ;
'ELEMENT' .and. trim(etype) = 'FIELD' OFF
?
? msg
wait to continue
loop
endif
enddo while floop
* program baktrev.prg
* purpose: to make queries in 'backwards' fashion, that is, with user
* providing object and the system finding what entity uses, contains,
* whatever, for the entity given
* programmer: al noel
*

```

\* files used: model, office, report, program, file, element, field, uses,  
 \* maintains, contains, inputs, outputs, calls

\* set exact on  
 SET FORMAT TO SCREEN

erase  
 set color to 30,14

set talk off  
 \* set up loop for work  
 store t to floop

do while floop

erase  
 \* init variables for screen

store , to git , to go:model  
 store , to go:report  
 store , to go:uses

store , to go:maintan  
 store , to go:office , to go:program  
 store , to go:file

store , to go:calls  
 store , to go:inputs  
 store , to go:outputs  
 store , to go:model , to go:element  
 store , to go:field

store , to go:contain  
 store , to go:file  
 store , to go:element  
 store , to go:contain

store , to go:element  
 store , to go:field

store , to go:contain  
 store , to go:file  
 store , to go:element  
 store , to go:contain

\* put up screen for user to give object and mark verb and subject  
 REMARK OBJECTS VERBS SUBJECTS  
 REMARK -----

5,3 say 'Model' get go:model  
 7,2 say 'Report' get go:report  
 5,31 say 'Is used by' get go:uses  
 7,25 say 'Is maintained by' get go:maintan  
 5,51 say 'Office' get go:office  
 9,1 say 'Program' get go:program  
 16,4 say 'File' get go:file  
 9,29 say 'Is called by' get go:calls  
 16,27 say 'Is inputted by' get go:inputs  
 11,26 say 'Is outputted by' get go:outputs  
 10,52 say 'Model' get go:model  
 12,1 say 'Element' get go:element

```

14,3 say 'Field' get go:field
13,26 say 'Is contained in' get gv:contain
12,53 say 'File' get gs:file
14,50 say 'Element' get qs:element
18,4 say 'To quit put an X in this field' get git
remark Enter the name of the entity besides its type and then mark 1 verb
remark field and 1 subject field. To process query move cursor down through
remark remaining fields using RETURN key.
*if git not blank , quit program and exit
if git <> ,
erase
release all
clear
return
else * load message to continue to variable
store 'Strike any key to continue ...' to msg
erase
* check for nonblank object variable, then, if not blank, use file indicated
** by verb variable marked with x and subject in same fashion
** load message about which query is being made according to selection and
** run display command loaded with values given .
if $(go:model,1,2) <> ' '
store 'The model' + go:model to query
if gv:uses <> ,
store query+ ' is used by the following offices ' to query
use uses
? query
?
? display ofc for trim(ename) = trim(go:model) .and. trim(etype) = ;
? 'MODEL', off
?
? msg
? wait to continue
loop
else
store query+ ' is maintained by the following offices ' to query
use maintains
? query
?
? display ofc,modelpoc for trim mdl = trim(go:model) off
?
? msg
? wait to continue
loop

```

```

endif
endif
if $(go:report,1,2) <> ' ' .and.    gv:uses    <> ' '
use uses
store 'The report '+go:report+ ' is used by the offices ' to query
? query
?
display ofc for trim(go:report) = trim(ename) .and. trim(etype) = ;
'REPORT', off
?
? msg
wait to continue
loop
endif
if $(go:report,1,2) <> ' ' .and. gv:outputs <> ' '
use outputs
store 'The report '+go:report+ ' is output by the models ' to query
? query
?
display ename1 for trim(qc:report) = trim(ename) .and. trim(etype1) = ;
'MODEL' .and. trim(etype) = 'REPORT', off
?
? msg
wait to continue
loop
endif
if $(gv:calls <> ' ' .and. qs:model <> ' '
if
use calls
STORE 'The program '+go:program+ ' is called by the models ' to query
? query
?
display ename1 for trim(go:program) = trim(ename) .and. trim(etype) = ;
'PROGRAM' .and. trim(etype1) = 'MODEL', off
?
? msg
wait to continue
loop
endif
endif
if $(go:file,1,2) <> ' '
store 'The file '+go:file to query
if gv:inputs
use inputs
store query+ ' is input by the following models ' to query
? query
?
endif

```



```

? display ename1 for trim(ename) = trim(qo:file) .and. trim(etype) = ;
? 'FILE'.and. trim(etype) = 'MODEL' off
?
? msg
? wait to continue
? loop
? endif
? if gv:outputs <> ' ' .and. gs:model <> ' '
? use outputs
? store query + ' is outputted to by the following models' to query
? query
?
? display ename1 for trim(ename) = trim(qo:file) .and. trim(etype) ;
? = 'FILE'.and. trim(etype1) = 'MODEL' off
?
? msg
? wait to continue
? loop
? endif
? endif
? if $(qo:element,1,2) <> ' '
? if gv:contain <> ' ' .and. gs:file <> ' '
? use contains
? store 'The element '+qo:element+ ' is contained by the files'to query
? query
?
? display ename1 for trim(ename) = trim(qo:element) .and. trim(etype) ;
? = 'ELEMENT'.and. trim(etype1) = 'FILE' off
?
? msg
? wait to continue
? loop
? endif
? endif
? if $(qo:field,1,2) <> ' '
? if gv:contain <> ' ' .and. gs:element <> ' '
? use contains
? store 'The field '+qo:field+ ' is contained by the elements ' to query
? query
?
? display ename1 for trim(ename) = trim(qo:field) .and. trim(etype) = ;
? 'FIELD'.and. trim(etype1) = 'ELEMENT' off
?
? msg
? wait to continue
? loop

```

```

endif
endif
endif
enddo while floop
* program foral.prg
** purpose: forward queries on aliases..user marks the type of entity
**           and the program then tells him all names and aliases in alias file
**           for that type of entity.
** programmer: al noel
** files used: alias
**
set exact on
SET FORMAT TO SCREEN
set color to 30,14
erase
use alias
set talk off
* set up loop for work
store t to fraLOOP
do while fraLOOP
  * init screen variables
  store , , to qmodel
  store , , to gprogram
  store , , to gfile
  store , , to goffice
  store , , to greport
  store , , to gfield
  store , , to qelement
  store , , to git
  * put screen up, for use to mark entity he wants aliases for
  7,11 say 'Model', get qmodel
  9,9 say 'Program', get gprogram
  11,10 say 'Report', get greport
  7,38 say 'File', get gfile
  9,35 say 'Element', get qelement
  11,37 say 'Field', get gfield
  13,15 say 'To quit', get qfield
  17,1 say 'Enter an X here', get git
  18,1 say 'Enter an X in the field besides the type of entity for which'
  read
  * if git not blank , quit program
  if git <>
    erase

```

```

clear
release all
return

else
* if q variable not blank then load type variable with appropriate type
* and load query variable with appropriate words for query message to use
*

if greport <> ' '
store 'The following reports have the associated aliases ' to query
store 'REPORT' TO TYPE
endif
IF QMODEL <> ' '
store 'The following models have the associated aliases ' to query
store 'MODEL' TO TYPE
ENDIF
IF QPROGRAM <> ' '
store 'The following programs have the associated aliases ' to query
store 'PROGRAM' TO TYPE
ENDIF
IF QOFFICE <> ' '
store 'The following offices have the associated aliases ' to query
store 'OFFICE' TO TYPE
ENDIF
IF QFILE <> ' '
store 'The following files have the associated aliases ' to query
store 'FILE' TO TYPE
ENDIF
IF QELEMENT <> ' '
store 'The following elements have the associated aliases ' to query
store 'ELEMENT' TO TYPE
ENDIF
IF QFIELD <> ' '
store 'The following fields have the associated aliases ' to query
store 'FIELD' TO TYPE
ENDIF
ERASE
* display query message built above and records that meet conditions
* loaded
? query
?
DISPLAY ENAME, ANAME FOR TRIM(ETYPE) = TYPE OFF
?
? 'Strike any key to continue...'
wait to continue
erase
endif
enddo while fraloop

```

```

* program bakal.prg
** purpose: backward queries on aliases. user gives the entity name and the
** program then tells him all aliases in alias file for the entity
** programmer: al noel
** files used: alias
**
set exact on
SET FORMAT TO SCREEN
set color to 30,14
erase alias
use alias
set talk off
* set up loop
store t to fialoop
do while fialoop
  * init screen variables
  store , to gmodel
  store , to qprogram
  store , to qfile
  store , ' to goffice
  store , ' to greport
  store , ' to gfield
  store , ' to gelement
  store , ' to git
  * put screen up for user to give entity he wants aliases for
7,11 say 'Model', get qmodel
9,9 say 'Program', get qprogram
11,10 say 'Report', get greport
7,38 say 'File', get qfile
9,35 say 'Element', get gelement
11,37 say 'Field', get gfield
13,15 say 'To quit enter an X here', get git
17,1 say 'Enter the name of the entity in the field beside its type '
18,1 say 'Provide only 1 name.'
read
* if git nonblank, quit program
  if git <> ,
    erase
    clear
    release all
    return
  else
    if q variable nonblank, load appropriate words to type variable and
    appropriate words to query variable for display command later

```

```

if $(qreport,3,1) <> ' '
store 'The report ' + qreport + ' has the following aliases ' to query
store 'REPORT' TO TYPE
store qreport to name
endif
IF $(QMODEL,3,1) <> ' '
store 'The model ' + QMODEL + ' has the following aliases ' to query
STORE 'MODEL' TO TYPE
store qmodel to name
ENDIF
IF $(QPROGRAM,3,1) <> ' '
store 'The program ' + QPROGRAM + ' has the following aliases ' to query
STORE 'PROGRAM' TO TYPE
store qprogram to name
ENDIF
IF $(QOFFICE,3,1) <> ' '
store 'The office ' + QOFFICE + ' has the following aliases ' to query
STORE 'OFFICE' TO TYPE
store qoffice to name
ENDIF
IF $(QFILE,3,1) <> ' '
store 'The file ' + QFILE + ' has the following aliases ' to query
STORE 'FILE' TO TYPE
store qfile to name
ENDIF
IF $(QELEMENT,3,1) <> ' '
store 'The element ' + QELEMENT + ' has the following aliases ' to query
STORE 'ELEMENT' TO TYPE
store qelement to name
ENDIF
IF $(QFIELD,3,1) <> ' '
store 'The field ' + QFIELD + ' has the following aliases ' to query
STORE 'FIELD' TO TYPE
store qfield to name
ENDIF
ERASE
* display query command built above and run actual display command
* then wait for user to hit any key to get fresh screen for another run
? query
?
DISPLAY          ANAME FOR TRIM(ETYPE) = TYPE .AND. TRIM(ENAME) = ;
TRIM(NAME) OFF
?
? 'Strike any key to continue...'
wait to continue
erase
endif

```

```

enddo while fraloop
** program foral.prg
** purpose: forward queries on categories. user marks the type of entity
**          and the program then tells him all names and associated categories
**          that type of entity
** programmer: al noel
** files used: category
**
set exact on
SET FORMAT TO SCREEN
set color to 30,14
erase
use category
set talk off
** set up loop
store t to fialoop
do * while fraloop
    store , , to gmodel
    store , , to qprogram
    store , , to gfile
    store , , to goffice
    store , , to greport
    store , , to gfield
    store , , to gelement
    store , , to git
    * put up screen
7,11 say 'Model', get gmodel
9,9 say 'Program', get qprogram
11,10 say 'Report', get greport
7,38 say 'File', get gfile
9,35 say 'Element', get gelement
11,37 say 'Field', get gfield
13,15 say 'To quit enter an X here' get git
17,1 say 'Enter an X in the field besides the type of entity for which'
18,1 say 'read'
* if git nonblank, quit program and return
    if git <> , ,
        erase
        clear
        release all
        return

```



```

else
* q variable marked dets. what gets loaded in type variable for use in
* display command and also what words go into query variable to tell user
* what he queried for along with output

if qreport <> ' '
STORE 'The following reports are in the associated category ' to query
store 'REPORT' TO TYPE
endif
IF QMODEL <> ' '
store 'The following models are in the associated category ' to query
STORE 'MODEL' TO TYPE
ENDIF
IF QPROGRAM <> ' '
store 'The following programs are in the associated category ' to query
STORE 'PROGRAM' TO TYPE
ENDIF
IF QOFFICE <> ' '
store 'The following offices are in the associated category ' to query
STORE 'OFFICE' TO TYPE
ENDIF
IF QFILE <> ' '
store 'The following files are in the associated category ' to query
STORE 'FILE' TO TYPE
ENDIF
IF QELEMENT <> ' '
store 'The following elements are in the associated category' to query
STORE 'ELEMENT' TO TYPE
ENDIF
IF QFIELD <> ' '
store 'The following fields are in the associated category ' to query
STORE 'FIELD' TO TYPE
ENDIF
ERASE
* display query variable words and actual results of retrieval, wait for use
* to strike any key to give him fresh screen for new query to run
? query
?
DISPLAY ENAME, CAT FOR THIM(ETYPE) = TYPE OFF
?
? 'Strike any key to continue...'
wait to continue
erase
endif
enddo while fraloop
* program bakcat.prg

```

\* purpose: backward queries on categories, user gives the entity name and the  
 \* program then tells him all categories in category fiel for the  
 \* entity

\* programmer: al noel

\* files used: category

set exact on  
 SET FORMAT TO SCREEN  
 set color to 30,14

erase

use category

set talk off

\* set up loop

store t to fralooop

do while fralooop

do \* init screen variables

store ' to gmodel

store ' to gprogram

store ' to gfile

store ' to goffice

store ' to greport

store ' to gfield

store ' to gelement

store ' to git

\* put screen up for user to give entity name to retrieve on

7,11 say 'Model' get gmodel

9,9 say 'Program' get gprogram

11,10 say 'Report' get greport

7,38 say 'File' get gfile

9,35 say 'Element' get gelement

11,37 say 'Field' get gfield

13,15 say 'To quit enter an X here' get git

17,1 say 'Enter the name of the entity in the field beside its type '

18,1 say 'provide only 1 name.'

read

\* if git nonblank, quit program and return

if git <>

erase

clear

release all

return

else

\* depending on q variable marked appropriate words are loaded into query  
 \* variable and to type variable to control display command later

```

if $(greport,3,1) <> ' '
store $(The report '+greport+' is in the associated category' to query
store 'REPORT' TO TYPE
store greport to name
endif
IF $(QMODEL,3,1) <> ' '
store $(The model '+QMODEL+' is in the associated category ' to query
store 'MODEL' TO TYPE
store qmodel to name
ENDIF
IF $(QPROGRAM,3,1) <> ' '
store $(The program '+QPROGRAM+' is in the associated category ' to query
store 'PROGRAM' TO TYPE
store qprogram to name
ENDIF
IF $(QOFFICE,3,1) <> ' '
store $(The office '+QOFFICE+' is in the associated category ' to query
store 'OFFICE' TO TYPE
store qoffice to name
ENDIF
IF $(QFILE,3,1) <> ' '
store $(The file '+QFILE+' is in the associated category ' to query
store 'FILE' TO TYPE
store qfile to name
ENDIF
IF $(QELEMENT,3,1) <> ' '
store $(The element '+QELEMENT+' is in the associated category' to query
store 'ELEMENT' TO TYPE
store qelement to name
ENDIF
IF $(QFIELD,3,1) <> ' '
store $(The field '+QFIELD+' is in the associated category ' to query
store 'FIELD' TO TYPE
store qfield to name
ENDIF
ERASE
* display query words built and run retrieval with display using type
* determined by what was loaded above from what was marked on screen
? query
?
DISPLAY CAT FOR TRIM(ETYPE) = TYPE .AND. TRIM(ENAME) = ;
TRIM(NAME) OFF
?
? 'Strike any key to continue...'
wait to continue
erase
endif
enddo while fraloop

```

```

* PROGRAM: REPT-MENU
* purpose: to permit user to select program which will provide printed
*          summary report
* programmer: al noel
* files used: model, office, report, file, program, element, uses, maintains
*             contains, calls, inputs, outputs,
SET FORMAT TO SCREEN
set color to 30,14
set talk off
ERASE
*set up loop
DO WHILE T
  1,0,TEXT
  This is the REPORT menu. It permits you to have the dictionary print
  summary type reports.

      REPORT MENU

  1) for a specific Model all users, maintainers, input& output files,
    and programs called (SUMOD)
  2) for a specific Report all users, models outputting
    (SUMREP)
  3) for a specific office all models used, all models maintained, and
    reports used. (SUMOFF)
  4) for a specific file all models it provides input to, receives output from,
    and elements it contains (SUMFIL)

    0 to quit

Enter the selection here
ENDTEXT
WAIT TO RMSELECT
  20,0 SAY
  21,0 SAY
*set up case statement for calling of program per user choice
DO CASE RMSELECT = 0
  USE
  clear
  ERASE
  RELEASE ALL

```

```

RETURN
CASE RMSELECT = 1
DO SUMOD
CASE RMSELECT = 2
DO SUMREP
CASE RMSELECT = 3
DO SUMOFF
CASE RMSELECT = 4
DO SUMFIL
* error message for wrong choice of numbers
OTHERWISE
ERASE
20,0 SAY Please enter values between 0 and 4 only
21,0 SAY Please try again
ENDCASE
ENDDO
RETURN

```

```

* PROGRAM SUREP.PRG
* * purpose: to produce printed report on report and all users, and models
* * output that report
* * programmer: al noel
* * files used: report, uses, outputs
SET TALK OFF
set color to 30,14
set format to screen
erase ' ' to space
store ' ' to space
* set up loop
store t to more
do while more
? Prompt user for name of report
? Please enter the unique name of the Report
accept Hit the space bar and RETURN to quit to rmod
erase
* if rmod blank, quit progra
if rmod = ' ' or rmod =
erase
clear
release all
return
else

```

```

* turn printer on
set print on
? The Report , rmod
?
? 'Is used by the following offices:'
* retrieve for users of report given by user of this program
?
use uses
display ofc for trim(ename) = trim(rmod) .and. trim(etype) = 'REPORT' ;
off
?
? 'Is produced by the following Models'
?
? retrieve for models that output this report
use outputs
display ename1 for trim(ename) = trim(rmod) .and. trim(etype1) = ;
'MODEL' .and. trim(etype) = 'REPORT', OFF
* prompt user to stike any key to make another run of program
?
? 'Strike any key to continue'
WAIT TO GOAHEAD
erase
set print off

```

```
endif
```

```

enddo
** PROGRAM SUMFIL.PRG
* * purpose: to produce printed report on a file and all models it is input by,
* * output to, and elements contained
* * programmer: al noel
* * files used: file, model, outputs, inputs, contains
SET TALK OFF
set color to 30,14
set format to screen
erase
store ' ' to space
store t to more

```



```

*set up loop
do while more
? Please enter the unique name of the File
accept Hit the space bar and RETURN to quit to rmod
erase
* if rmod blank, quit program
if rmod = ' '.of. rmod =
erase
clear
release all
return
else
* turn printer on
set print on
? The File , rmod
? ? ? ? ? Provides input to the following Models:
? ? ? ? ? print out that which is for file given by user
* use inputs
display ename1 for trim(rmod) = trim(ename) .and. trim(etype) = 'FILE' ;
.and. trim(etype1) = 'MODEL' off
? ? ? ? ? Accepts output from the following Models:
? ? ? ? ? use outputs
display ename1 for trim(rmod) = trim(ename).and. trim(etype) = 'FILE' ;
.and. trim(etype1) = 'MODEL' off
? ? ? ? ? Contains the following Elements
? ? ? ? ? use contains
display ename1 for trim(ename1) = trim(rmod) .and. trim(etype) = ;
'ELEMENT'.AND. trim(etype1) = 'FILE' off
? ? ? ? ? * wait for user to strike any key to give fresh screen to run another
* report
? ? ? ? ? 'Strike any key to continue'
WAIT TO GOAHEAD
erase
* turn printer off , turn it back on up in loop
set print off
SET TALK Off
endif

```

enddo

```

* PROGRAM SUMOD.PRG
** purpose: to produce printed report on model and all users, maintainers,
**          input, output files, and programs called
**
** programmer: al noel
**
** files used: model, uses, maintains, inputs, outputs, calls
**
SET TALK OFF
set color to 30,14
set format to screen
erase
store ' ' to space
** set up loop
store t to more
do while more
? Please enter the unique name of the Model
* prompt for model name
accept Hit the space bar and RETURN to quit to rmod
erase
* if rmod blank, quit program
if rmod = ' ' .or. rmod =
erase
clear
release all
return
else
* turn printer on
set print on
? The Model , rmod
?
? 'Is used by the following offices:'
?
* retirev users of mode
use uses
display ofc for trim(ename) = trim(rmod) .and. trim(etype) = 'MODEL' ;
off
?
? 'Is maintained by the following offices with the point of contact:'
?
* retrieve maintainers of model
use maintains
display ofc,modelpoc for trim(mdl) = trim(rmod) off
?
? Inputs the following files:
?
* retrieve files input by model

```

```

use inputs
display ename for trim(ename1) = trim(rmod) .and. trim(etype1) = ;
MODEL .and. trim(etype) = 'FILE' Off
? ? 'Outputs to the following files:'
? ? * retrieve files output to by model
use outputs
display ename for trim(ename1) = trim(rmod) .and. trim(etype1) = ;
MODEL .and. trim(etype) = 'FILE' Off
? ? * Calls the following programs
? ? * retrieve programs called
use calls
display ename for trim(ename1) = trim(rmod) .and. trim(etype1) = ;
MODEL .and. trim(etype) = 'PROGRAM' Off
? ? * let user hit any key to get fresh prompt to run program again
? ? * Strike any key to continue'
WAIT TO GOAHEAD
erase
* turn printer off , it will be set back on above
set print off
SET TALK OFF
endif

```

```

enddo
** PROGRAM SUMOFF.PRG
* * * purpose: to produce printed report on a office and all models used,
* * * all models maintained, and reports used
* * * programmer: al noel
* * * files used: office, uses, maintains
SET TALK OFF
set color to 30,14
set format to screen
erase ' ' to space
store ' ' to space
* set up loop
store t to more
do while more
? prompt user for office symbol to use
? please enter the Office Symbol of the Office

```

```

accept Hit the space bar and RETURN to quit to rmod
erase
* in rmod blank, quit program
if rmod = ' ' .or. rmod =
  erase
  clear
  release all
  return
else
* turn printer on
set print on
? The Office , rmod
? ? ? 'Uses the following Models:'
? ? ? * retrieve for models used by office
? ? ? use uses
? ? ? display ename for trim(ofc) = trim(rmod) .and. trim(etype) = 'MODEL';
off
? ? ? 'Uses the following Reports:'
? ? ? * retrieve for reports used by office
? ? ? display ename for trim(ofc) = trim(rmod) .and. trim(etype) = 'REPORT' off
? ? ? 'Maintains the following Models:'
? ? ? * retrieve on models maintained
? ? ? use maintains
? ? ? display mdl      for trim(ofc) = trim(rmod)      off
? ? ? ? ?
? ? ? ? ? 'Strike any key to continue'
? ? ? * prompt user for a strike of any key to go ahead with new prompt and run
? ? ? * of program
? ? ? WAIT TO GOAHEAD
? ? ? erase
? ? ? set print off
? ? ? SET TALK OFF
? ? ? endif
enddo

```

## LIST OF REFERENCES

1. Pressman, R. S., Software Engineering: A Practitioner's Approach, p. 95, McGraw Hill, 1982.
2. Wasserman, A. I. and Shewmake, D. T., "Rapid Prototyping of Interactive Information Systems", ACM SIGSOFT Software Engineering Notes, v. 7, p. 177, December 1982.
3. Martin, J., Application Development Without Programmers, Prentice Hall, p. 59, 1982.
4. Bell, M.D., "Penny Wise Approach to Data Processing", Harvard Business Review, v. 59, p. 111, September 1982.
5. Carey, T.T. and Mason, R.E.A., "Prototyping Interactive Information Systems", Communications of the ACM, v. 26, p. 347, May 1983.
6. Blum, B. I., "Rapid Prototyping of Information Management Systems", ACM SIGSOFT Software Engineering Notes, v. 7, p. 35, December 1982.
7. Pressman, R. S., Software Engineering: A Practitioner's Approach, p. 132, McGraw Hill, 1982.
8. Martin, J., Application Development Without Programmers, p. 82, Prentice Hall, 1982.
9. Blum, B. I., "Rapid Prototyping of Information Management Systems", ACM SIGSOFT Software Engineering Notes, v. 7, p. 35, December 1982.
10. Pressman, R. S., Software Engineering: A Practitioner's Approach, p. 338, McGraw Hill, 1982.
11. Blum, B. I., "Still More About Rapid Prototyping", ACM SIGSOFT Software Engineering Notes, v. 8, p. 9, July 1983.
12. Jackson, M. A. and McCracken, D. D., "Life Cycle Considered Harmful", ACM SIGSOFT Software Engineering Notes, v. 7, p. 31, April 1982.
13. Carey, T.T. and Masch, R.E.A., "Prototyping Interactive Information Systems", Communications of the ACM, v. 26, p. 352, May 1983.



14. Kovacs, A. and Meyer, K., "Model Systems", Datamation (Reader's Forum), v. 29, p. 248, September 1983.
15. Wasserman, A. I. and Shewmake, D. T., "Rapid Prototyping of Interactive Information Systems", ACM SIGSOFT Software Engineering Notes, v. 7, p. 175, December 1982.
16. Carey, T.T. and Mason, R.E.A. "Informations Systems Prototyping: Techniques, Tools and Methodologies", Infor, v. 21, p. 183, August 1983.
17. Ibid, p. 184.
18. Appleton, D. S., "Data Driven Prototyping", Datamation, v. 29, p. 252, November 1983.
19. McNurlin, B. C., "Developing Systems by Prototyping", EDP Analyzer, v. 19, p. 5, September 1981.
20. Wasserman, A. I., "Introduction to System Design Methodology", Tutorial on Software Design, p. 43, IEEE Computer Society Press, 1983.
21. McNurlin, B. C., "Developing Systems by Prototyping", EDP Analyzer, v. 19, p. 5, September 1981.
22. Ibid, p. 3.
23. Wiorkowski, G., "The Use of Relational DBMS For Prototyping", Computerworld, v. 19, p. 20, October 1983.
24. Lusa, J.M., "Prototyping Maxi Applications on Micros", Infosystems, v. 9, p. 90, September 1981.
25. Carey, T.T. and Mason, R.E.A., "Prototyping Interactive Information Systems", Communications of the ACM, v. 26, p. 349, May 1983.
26. Appleton, D. S., "Data Driven Prototyping", Datamation, v. 29, p. 254, November 1983.
27. Van Duyn, J., Developing A Data Dictionary System, p. 2, Prentice Hall, 1981.
28. Kovacs, A. and Meyer, K., "Model Systems", Datamation (Reader's Forum), v. 29, p. 252, September 1983.
29. King, J. L., "Centralized versus Decentralized Computing", Computing Surveys, v. 15, p. 334, December 1983.



30. Kovacs, A. and Meyer, K., "Model Systems", Datamation (Reader's Forum), v. 29, p. 250, September, 1983.
31. Martin, J. Application Development Without Programmers, pp. 216-217, Prentice Hall, 1982.
32. Weldon, J. L., Data Base Administration, p. 176, Plenum Press, 1981.
33. Ibid, p. 176.
34. Leong-Heong, B. W. and Plagman, B. K., Data Dictionary/Directory Systems, p. 77, Wiley, 1982.
35. Ibid, p. 78.
36. Ibid, p. 78.
37. Ibid, pp. 78-79.
38. Ibid, p. 79.
39. Ibid, p. 79.
40. Pressman, R. S., Software Engineering: A Practitioner's Approach, p. 200, McGraw Hill, 1982.
41. Ibid, p. 200.
42. Leong-Heong, B. W. and Plagman, B. K., Data Dictionary/Directory Systems, p. 76, Wiley, 1982.
43. Ibid, pp. 80-86.
44. Zaslow, M., NSA/CSS Manual 81-3 Software Product Standards, p. III-50, National Security Agency, 1979.
45. Weinberg, V., Structured Analysis, p. 156, Yourdon Press, 1978.
46. Fry, J.P. and Teorey, T.J., Class Notes NSA Course MP-4H4 Principles of Data Base Management, p. E15, 1982.
47. Windsor, A. T., Using the ICL Data Dictionary: Proceedings of the User Group, p. 151, Shiva Publishing Unlimited, 1980.

48. Curtice, R. M., "Data Dictionaries: An Assessment of Current Practices and Problems", Proceedings Very Large Data Bases Seventh International Conference Zaniolo, C. and Delobel, C., eds., p. 565., IEEE Computer Society Press, 1981.

# INITIAL DISTRIBUTION LIST

	No.	Copies
1. Defense Technical Information Center Cameron Station, Alexandria, Virginia 22314		2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93943		2
3. Department Chairman, Code 54 Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943		1
4. Dr. Dan Dolk, Code 54Dk Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943		1
5. Ccmmander US Army MILPERCEN ATTN: DAPC-PLS 200 Stovall Street Alexandria, VA 22332		1
6. Major Alan F. Noel Survey Section SHAPE APO NY 09055		2
7. Computer Technology Program Code 37 Naval Postgraduate School Monterey, California 93943		1





2413

Thesis  
N6587  
c.1

No

Thesis  
N6587  
c.1

Noel

Prototyping with  
data dictionaries for  
requirements analysis.

22  
30

22 APR 87	3 2 2 1 1
30 JUN 87	3 1 7 7 5
24 JAN 88	3 5 2 3 5
10 FEB 88	3 7 2 1 1
30 MAR 88	3 6 3 4 4

2413

Thesis  
N6587  
c.1

Noel

Prototyping with  
data dictionaries for  
requirements analysis.





thesN6587

Prototyping with data dictionaries for r



3 2768 000 61073 7

DUDLEY KNOX LIBRARY